USER GUIDE

for the eBook system developed as part of the Stat-JR software package

School of Electronics and Computer Science
University of Southampton

Revision Sheet

Release No.	Date	Revision Description
v1.0	23/09/2011	First draft.
v2.0	13/10/2011	Updated version based on the comments from Danius.
v2.1	20/10/2011	An important correction made in Section 4.4.4
v2.2	23/10/2011	Updated version based on the comments from Richard.
v2.3	27/10/2011	Added latex as an option of static content.
v2.4	04/01/2012	Having multiple templates in the same Activity Region; Adding dataset summary as a dynamic output without template run; Adding template input set as a separate file; Attaching resource files (template/dataset) to eBook file; Added bottom page number list; Added page number input box; Added reloading function in eBook reading interface for authoring; Added eBook filechecker function and its integration.
v2.5	21/03/2012	Updated version based on the comments from Bill. New style and linked table of contents, consistent page numbering.
V2.6	30/03/2012	Updated based on comments and questions from Camille; Updated figures and text for interface changes (of menu and rsrc veiw); Added hideon/showon attribute; Added data access in tabular type output with XPath expressions; Added uploading user's own dataset; Added showing only selected rows/columns in tabular output; Added appendix of a list of markers introduced in eBook; Added authoring tab in resource view;
V2.7	10/04/2012	Updated based on comments from Danius
V2.7.1	11/04/2012	Revised structure for section 4.4, and other minor updates;

USER GUIDE

TABLE OF CONTENTS

1 GENERAL INFORMATION 2

1.1 eBook System Overview 2

1.2 About this User Guide 2

1.3 User Levels 2

2 SYSTEM INSTALLATION 4

2.1 System Installation 4

2.1.1 Pre-installation requirements 4

2.2 System Configuration 5

3 GETTING STARTED AS AN EBOOK READER 7

3.1 eBook System Main page 7

3.1.1 Importing an eBook 8

3.1.2 Creating a Reading Process 11

3.1.3 Resuming an Existing Reading Process 12

3.2 eBook Reading Interface 12

3.2.1 System Menu 13

3.2.2 Reading Area 14

3.2.3 Reading an eBook with Dynamic Content 15

3.2.4 Regenerating the dynamic content with a different answer to an input question 16

3.2.5 Regenerating the dynamic content with your own dataset 17

3.2.6 Table of contents 18

3.3 eBook Resource View 19

3.3.1 Resource Tree view 20

3.3.2 The Information area 21

3.3.3 Navigation in the Resource View 22

3.3.4 Exporting a resource 22

3.3.5 Reversing the eBook content to a previous state 23

4 ADVANCED GUIDE FOR EBOOK AUTHORS 27

4.1 The eBook - an inside view 27

4.1.1 File Structure 27

4.1.2 Organization of Resources 27

4.2 Preparing the Resources 30

4.2.1 Template and Dataset File 30

4.2.2 Template Input File 30

4.2.3 Resource File Location 31

4.3 ebookdef.n3 – Defining the Resource Organisation 31

4.4 ebookpages.html – Writing the Content 36

4.4.1 Adding Static Content - Text, Images and LaTeX 36

4.4.2 Making Chapters and Pages 37
4.4.3 Chapter and Section Headers 38
4.4.4 Adding Dynamic Content 39
4.4.5 Accessing and configuring dynamic output 41
4.4.6 Show/Hide static element with dynamic content 43

4.5 Authoring Tools 44
4.5.1 eBook Reloading Tool 44
4.5.2 The eBook Filechecker 46
4.5.3. The Authoring Tab in Resource View 50

5 APPENDIX 52

Appendix I. 'ebookdef.n3' file for the example eBook described in Section 4.1 52

Appendix II. An example 'ebookpages.n3' file 54

Appendix III. List of HTML Markers Introduced in eBook System 57

1 GENERAL INFORMATION

1 GENERAL INFORMATION

1.1 eBook System Overview

The purpose of the eBook system, one of the many features of the Stat-JR software package, which is created as part of the ESRC-funded e-STAT project, is to embed the executable software packages used by Stat-JR within traditional notebooks, thus producing an interactive reading experience. The resulting executable books combine the narrative advantage of traditional books with the experimental and interactive advantages of software packages and provide an improved and more user-friendly experience.

1.2 About this User Guide

This User Guide provides step-by-step instructions on how to import and read them, whilst more advanced users interested in creating their own eBooks will find an ‘under-the-bonnet’ description of how to author their own.

1.3 User Levels

- eBook reader

An eBook reader is a user who is interested in browsing the content of certain eBooks. eBook readers use the system to import and read eBook files. Note that 'reading' refers to a process which is a bit more interactive than implied by the common use of the word. For example, as a user ‘reads’ through an eBook, he or she will change its content by making choices when prompted to do so (for instance by selecting particular variables to analyse).

- eBook author

An eBook author is an advanced user who is an expert or specialist in a field and would like to share his/her knowledge via eBooks. They understand the internal structure of eBooks, and can use proper tools to create and edit an eBook file.

2 SYSTEM INSTALLATION

2 SYSTEM INSTALLATION

2.1 System Installation

This section is for users who want to install the Stat-JR software package manually. If you have an all-in-one version of Stat-JR software, please refer to the guide attached with it.

2.1.1 Pre-installation requirements

Python 2.7

Most parts of the eBook system are developed with Python. Thus the Python interpreter is the essential software required to run the eBook system.

Python accompanying packages for Stat-JR

Several Python accompanying packages are required by the Stat-JR module:

- NumPy
- SciPy
- Numexpr
- Matplotlib
- Setuptools
- Mako 0.4.2 or later
- datetime

Python accompanying packages for Core/Middleware module

The following Python accompanying package(s) is(are) required by the Core/Middleware module:

- Web.py

Python accompanying package for Resource management module

The following Python accompanying package(s) is(are) required by the Core/Middleware module:

- rdflib
- rdfextras

.NET framework for Stat-JR module

Microsoft .NET framework is required if you want to use the execution engine Stat-JR, which currently has some modules implemented in VB.NET. This requirement may be removed after the tool is migrated to Python.

2.2 System Configuration

You need to correctly configure the paths of various execution engines in the 'setting.cfg' file which is found in the root directory of the eBook system folder. The content of the 'setting.cfg' file is as follows:

```
[Global]
data_path=..\..\..\datasets

[MLwiN]
executable=C:\Program Files (x86)\MLwiN v2.22\mlwin.exe

[Demo]
executable=C:\StatJR\Demo\bin\Demo.exe
working_directory=C:\StatJR\Demo\bin

[WinBUGS]
executable=C:\WinBUGS14\WinBUGS14.exe
working_directory=C:\WinBUGS14

[STATA]
executable=C:\Program Files (x86)\Stata11\StataMP-64.exe

[R]
executable=C:\Program Files\R\R-2.12.0\bin\R.exe
```

Do not change the path under the [Global] category. For all the other categories, change the value of the 'executable' to the correct path: i.e. where the corresponding software package executable is installed if it differs on your machine. Note that, at the moment, the algebra system executable for the StatJR engine is placed under the category '[Demo]'.

3 GETTING STARTED AS AN EBOOK READER

3 GETTING STARTED AS AN EBOOK READER

This section provides a guide for eBook users on how to perform general eBook reading using the eBook system interface and widgets provided.

3.1 eBook System Main page

The first page shown when the eBook system is loaded is the main system page, where users can choose an eBook to be read. The system allows an eBook to be read multiple times, and the information associated with each reading is called a reading process. On the main page users can choose the reading process with which they would like to start. Currently, the following functionality is supported on the main page:

- Import an eBook
- Display general information about an eBook
- Create a reading process for an eBook
- Resume an existing eBook reading process

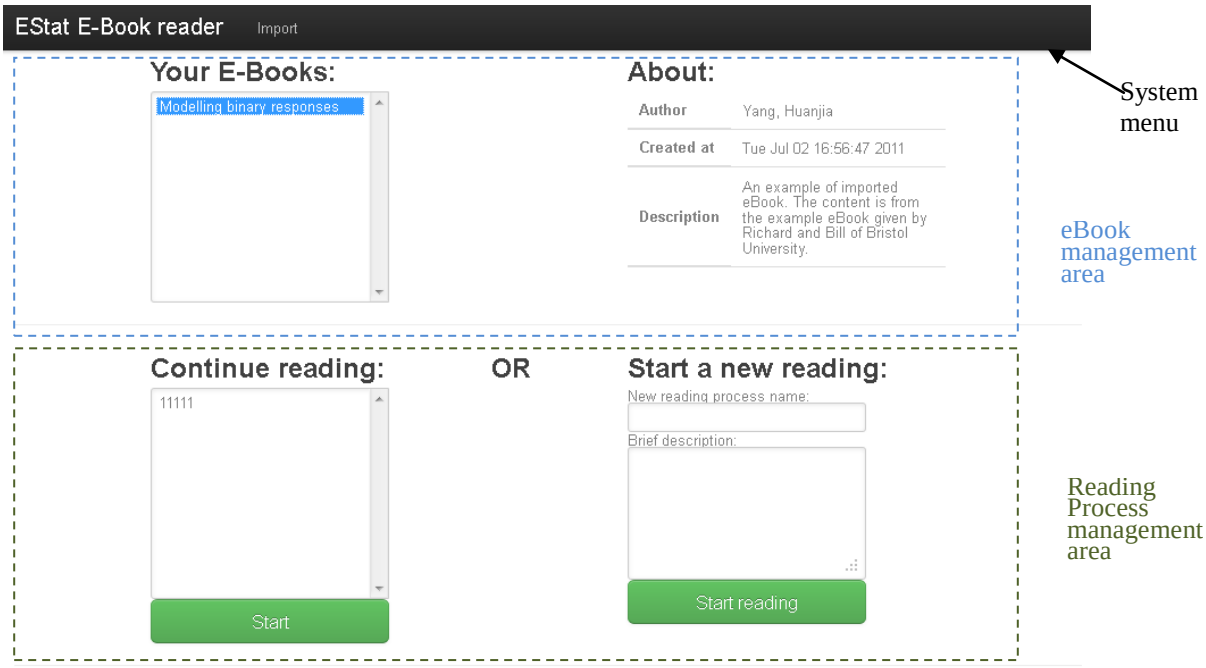


Figure 3-1. eBook system main page

3.1.1 Importing an eBook

As an eBook reader you will usually have access to one or more eBooks, each in the form of a single zip file saved on your hard disk or flash drive. You need to import the zip file of an eBook into the eBook system before it can be read.

To import an eBook file, click on the button ‘Import’ on the system menu (black bar at the top) of the eBook system main page.

Figure 3-2. Import button

A dialogue box will be opened for you to specify the location of the eBook file on your disk. Currently the eBook system can only accept an eBook file saved on your hard disk, flash drive or network drives.

Figure 3-3. eBook import dialogue

After selecting the correct eBook file, press the green import button in the lower right corner of the dialogue box. The box will then close and the system will perform a file check for the content in the eBook file. A page with the result showing possible error and warning information will be displayed as soon as the checking is finished. As shown in Figure 3-4, you will need to select whether to proceed with the import at this stage. If you press "Import", the system will proceed. If you press "Return", the system will abandon the import process and go back to the main page. Please refer to Section 4.5.2 for details of the file checking function and the error/warning messages.

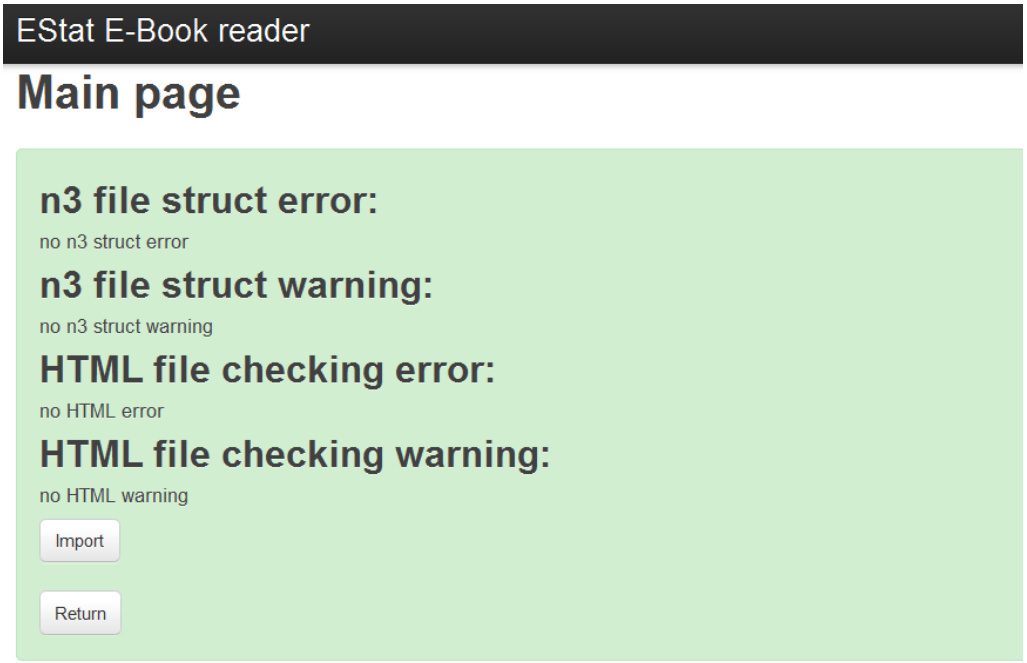


Figure 3-4. File checking result page

If you have selected to proceed in the previous step, the system will import the eBook file and a message about the import result will be displayed. If the eBook file has been imported successfully, a green 'success' message, as shown in Figure 3-5, will be displayed.

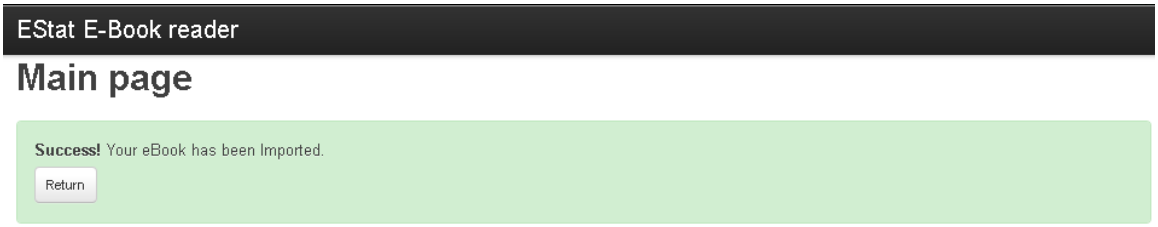


Figure 3-5. eBook import status page - success

An eBook file can only be imported once into the system. When you attempt to import an eBook file that has already been imported in the system, a blue 'eBook exists' error message, as shown in Figure 3-6, will be displayed.

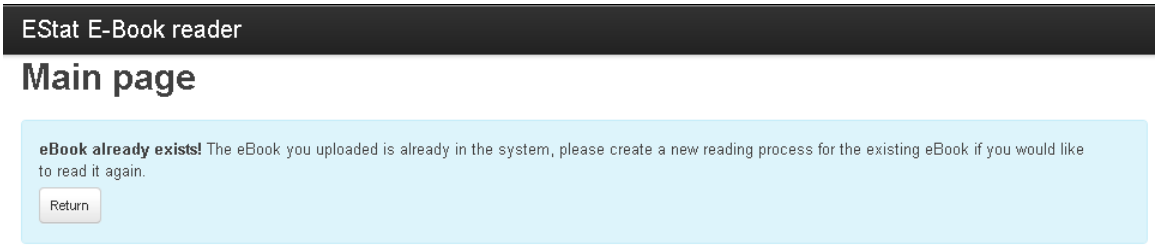

EStat E-Book reader
Main page
eBook already exists! The eBook you uploaded is already in the system, please create a new reading process for the existing eBook if you would like to read it again.
Return

Figure 3-6. eBook import status page – eBook exists

If the eBook file you are importing is broken or is not a valid eBook file, a red 'failed' error message will be displayed, as shown in Figure 3-7.

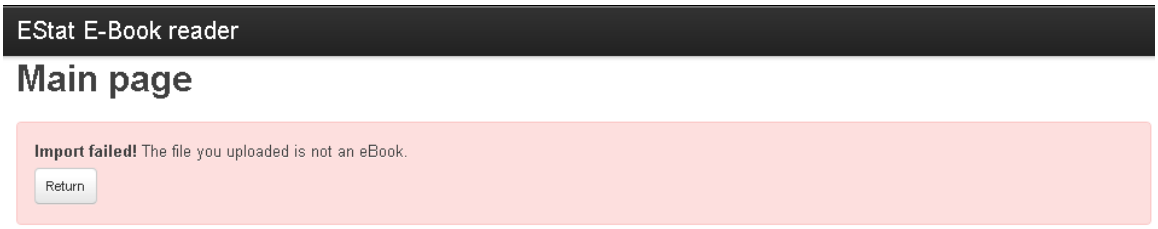


Figure 3-7. eBook import status page – failed

After the import status message has been displayed, press the 'return' button to go back to the system main page. If your eBook file has been imported successfully, its title should now appear in the 'Your E-Books' list located on the left of eBook management area. If you now click on the eBook's title in this list, the author, creation date, and a short description, will be displayed on the right-hand side.

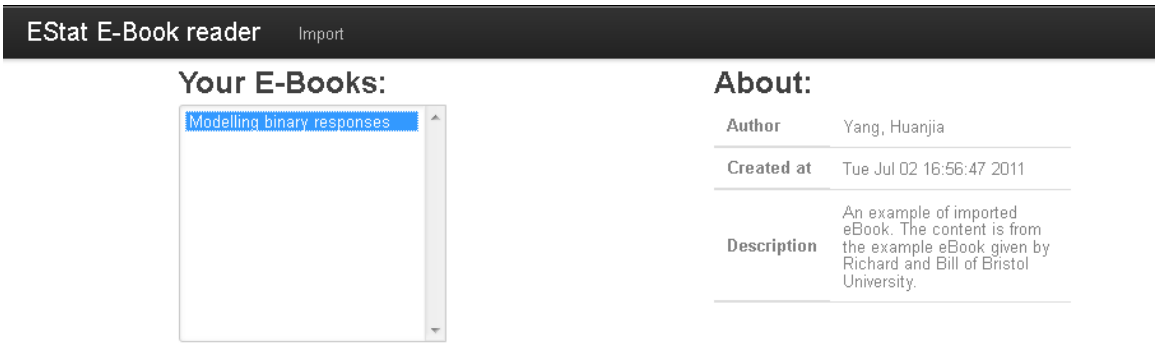


Figure 3-8. Selecting an imported eBook

3.1.2 Creating a Reading Process

An eBook can be read for multiple times by different users. For each eBook, the system deems the procedure and the inputs of a reader's reading as a 'reading process'. By doing this it allows each user to save and resume their own reading.

Figure 3-9. Creating a new reading process

To read an eBook you must first specify a reading process. You can do this by either creating a new reading process, or by selecting an existing one. For selecting an existing one please refer to the next section. To create a reading process for a particular eBook, click and select the eBook you want to read in the eBook select box on the main page, as shown in Figure 3-9. Then go to the 'Start a new reading' area located towards the lower right of the page, provide a name and, if you want, a brief description of the new reading process, and press the green 'Start reading' button to start reading your eBook, as shown in Figure 3-9. Since you might want to read an eBook multiple times, you can create as many reading processes as you like for each eBook in the system.

3.1.3 Resuming an Existing Reading Process

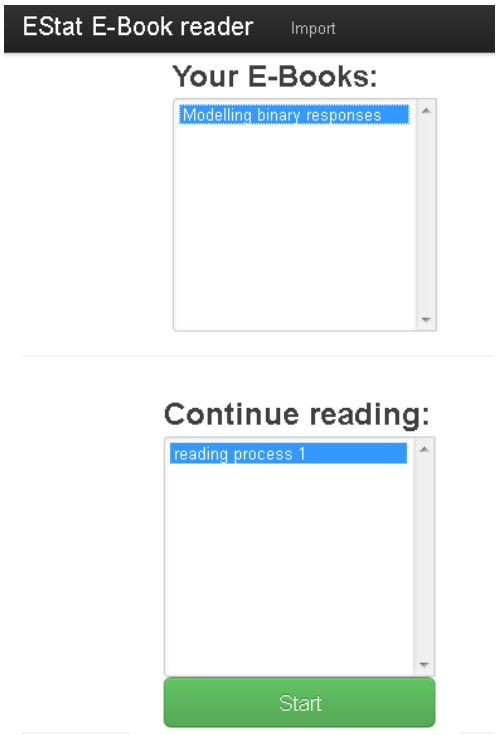


Figure 3-10. Resuming an existing reading process

The status of your reading is automatically recorded by the system during your reading process. You can always close/leave the system or go back to the main page in the middle of your reading, and come back later to resume the reading process from where you stopped. To resume a reading process, click and select the eBook; the title of the existing reading processes for the eBook you have selected will then be displayed in the select box below. Select the reading process you want to resume and press the green 'Start' button to start reading again.

3.2 eBook Reading Interface

After you create and start a new reading process, the eBook system will load the reading interface with the content of your selected eBook. If it is a new reading process (i.e. if you are not resuming a saved one) the eBook will be displayed from the first page. Figure 3-11 shows the general reading interface of the eBook system.

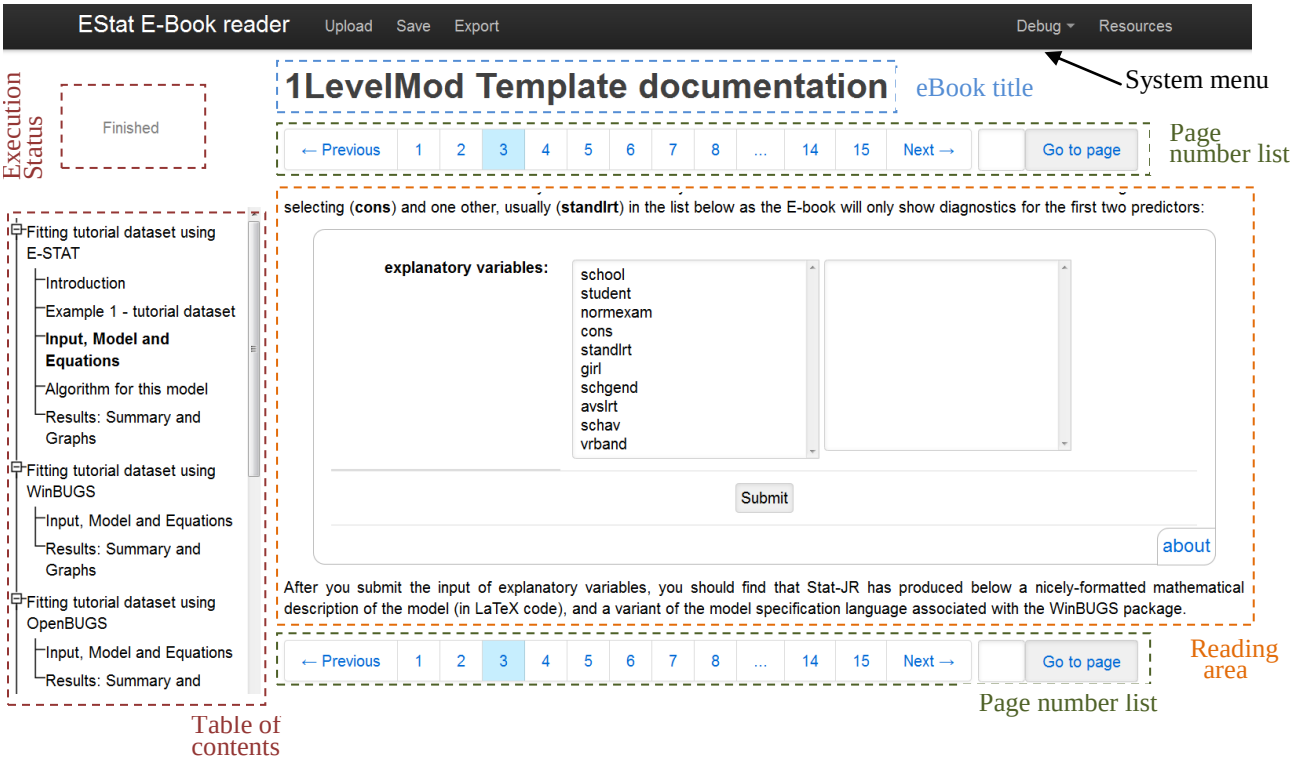


Figure 3-11. eBook reading interface

3.2.1 System Menu

After you enter the reading interface, more options will be available on the system menu (see Figure 3-12) .

Figure 3-12. System menu of reading interface

The 'Resources' button, on the right of the system menu, opens a resource view with information about the current eBook executions running behind the scene. Further details about the resource view will be provided in Section 3.3. Currently, the 'Save' and 'Export' buttons are not supported.

3.2.2 Reading Area

The majority of the reading interface is occupied by the reading area. The contents of the eBook are displayed within this reading area. You can scroll or navigate within the content area.

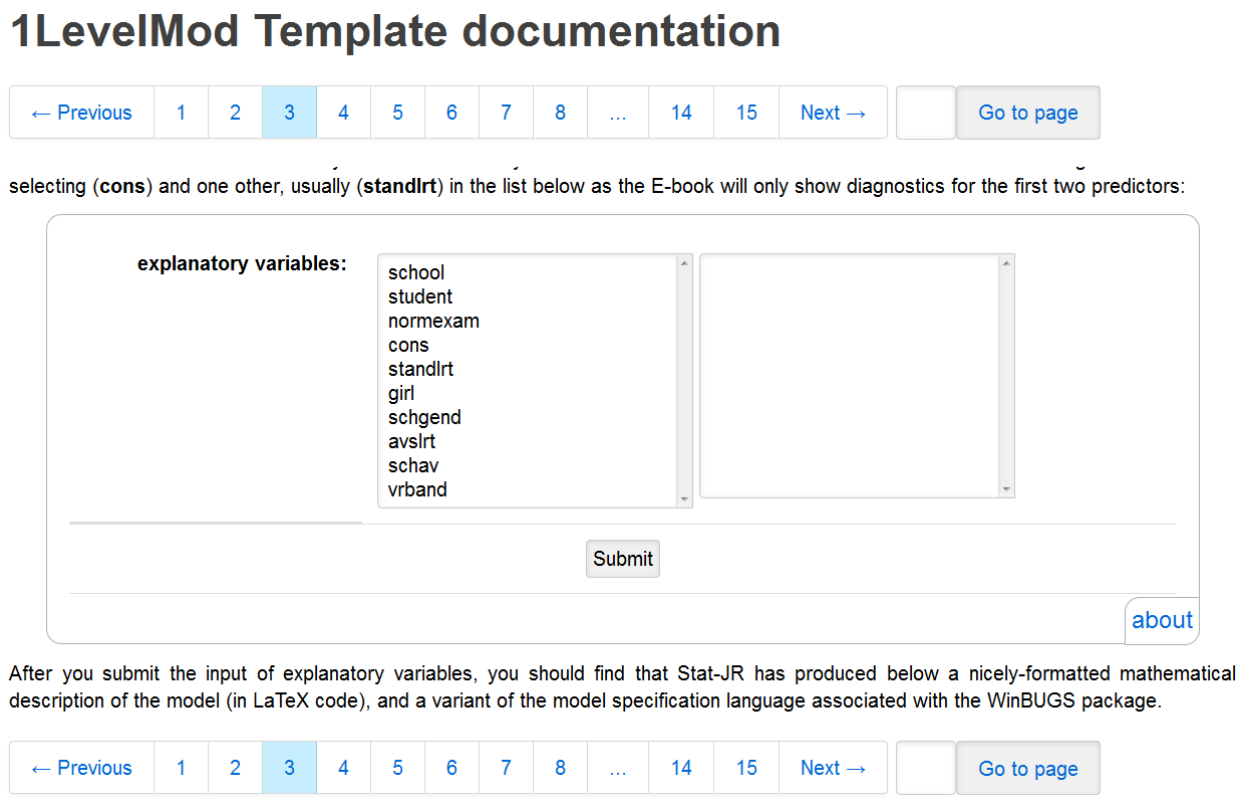

1LevelMod Template documentation
← Previous 1 2 3 4 5 6 7 8 ... 14 15 Next →
Go to page
selecting (cons) and one other, usually (standlrt) in the list below as the E-book will only show diagnostics for the first two predictors:
explanatory variables:
school
student
normexam
cons
standlrt
girl
schgend
avslrt
schav
vrband
Submit
about
After you submit the input of explanatory variables, you should find that Stat-JR has produced below a nicely-formatted mathematical description of the model (in LaTeX code), and a variant of the model specification language associated with the WinBUGS package.
← Previous 1 2 3 4 5 6 7 8 ... 14 15 Next →
Go to page

Figure 3-13. The main reading area, with the title and page number list blocks

At the top and bottom of the reading area you will find the eBook title and also the page number list. The page number list block displays a list of pages contained in the eBook, with the current page highlighted in light blue. You can click on a page number to jump to it, or you can used the ‘previous’ and ‘next’ button to move to adjacent pages. For long eBooks, the page number list will only show a few page numbers, but there is also a page number input box at the end of the list, where you can enter the page number of the page that you want to jump to (e.g. if not otherwise shown in the list).

3.2.3 Reading an eBook with Dynamic Content

EBook contains dynamic content, which are results generated by the executions defined in the eBook. They are generated during eBook reading based on input given by both author and user, and are inserted and rendered to the user when they become available. In general, the dynamic content can be categorised into two distinct groups: input questions and execution results.

An input question can be a text box, a drop list or a select box allowing you to specify certain parameter values required by the executions running behind the eBook. Figure 3-14 shows an example in which an input question is displayed in the form of select boxes.

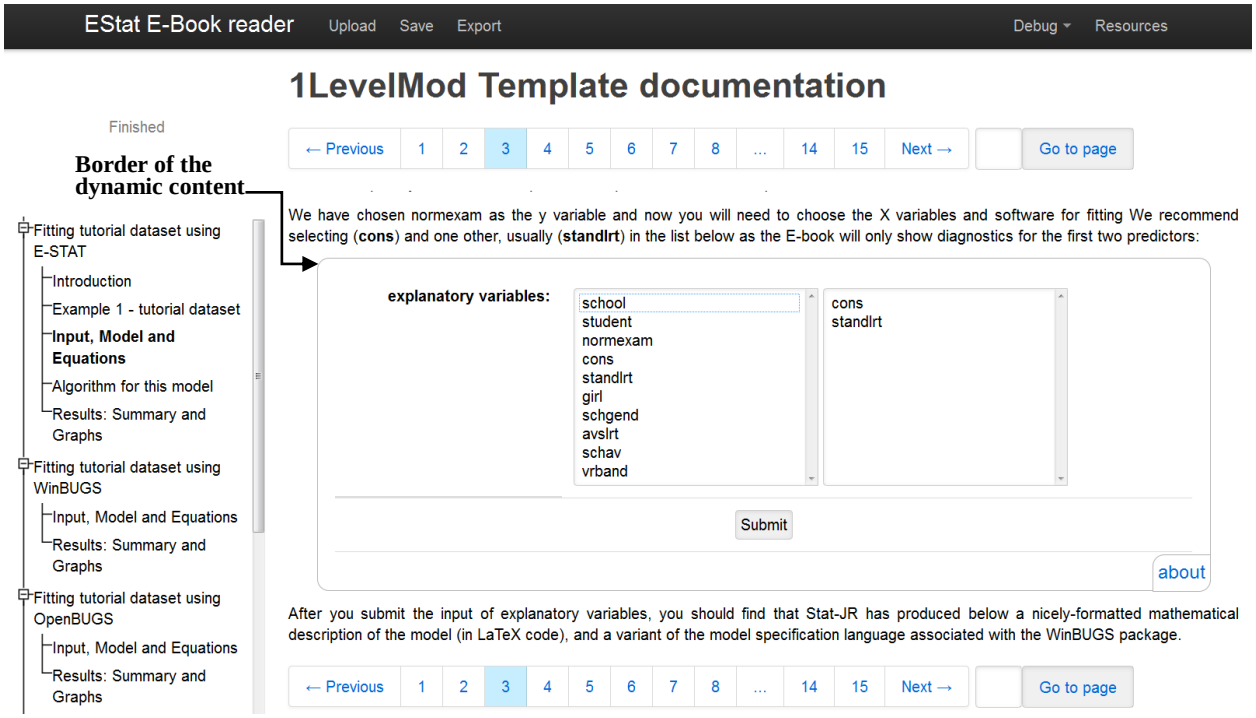


Figure 3-14. An input question in the form of a select box

The execution results could be graphs, tables, texts, mathematical equations and codes. They do not usually consist of interactive elements, and are simply inserted when they become available. Figure 3-14 shows an example where dynamic content, in the form of an equation, is displayed after the reader has responded to the parameter request in Figure 3-14; if the reader had provided different input (in this case different explanatory variables), then the dynamic content would be different.

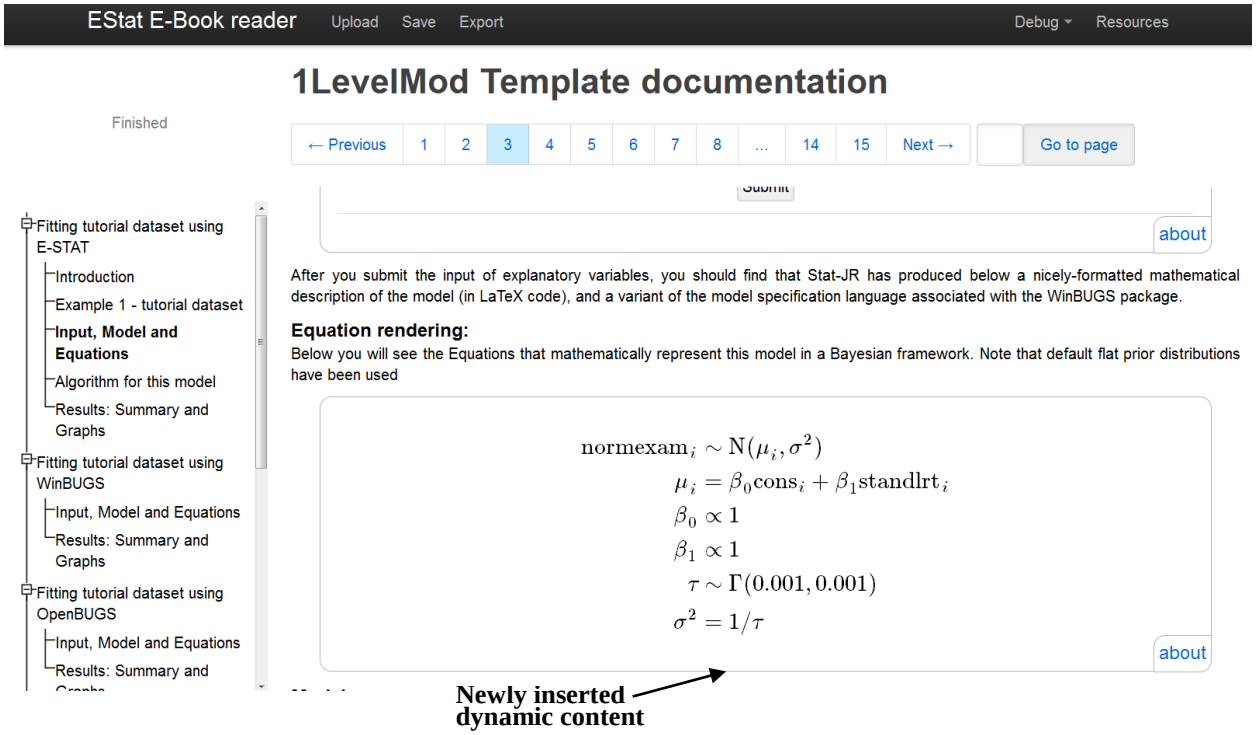


Figure 3-15. An equation inserted as dynamic content

3.2.4 Regenerating the dynamic content with a different answer to an input question

After reading the dynamic content, you can specify a different parameter value and the eBook will regenerate dynamic content to reflect your new input. To do this, go back to the input question, select or enter a different value, and then click "Submit". The eBook will then regenerate and update all the related dynamic content.

For example, if we were to go back in the page, select ‘vrband’ instead of ‘standlrt’ (see Figure 3-15) as an explanatory variable (still keeping 'cons') and then press submit, the dynamic content will be updated based on this new input, as shown in Figure 3-16.

You can repeat this procedure multiple times. The dynamic content in the eBook will always be updated with results based on the last input value you submitted.

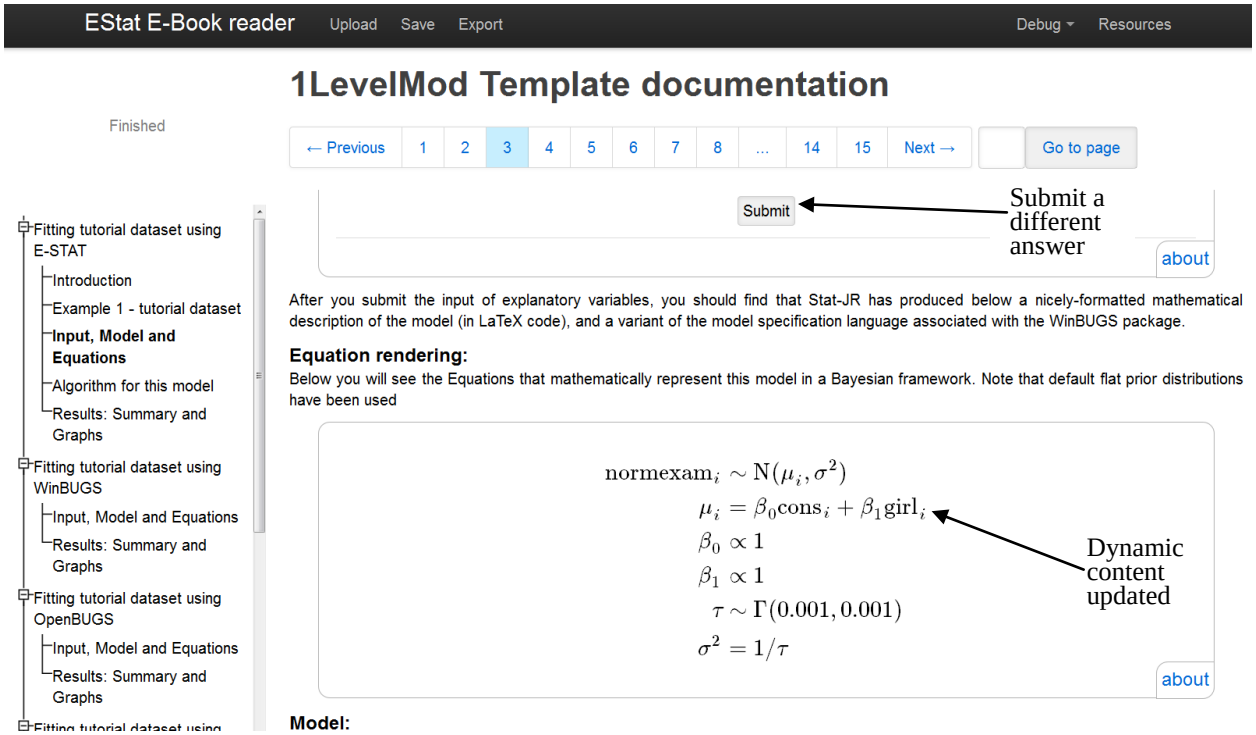


Figure 3-16. Regenerating the dynamic content with a different answer to an input question

3.2.5 Regenerating the dynamic content with your own dataset

After reading the dynamic content, you can regenerate them with your own dataset. You will need to upload the dataset and the eBook will regenerate dynamic content with it. To do this, go to the System Menu of the reading interface and click on the "upload" button. The system will open a dataset uploading dialogue as shown in Figure 3-17.

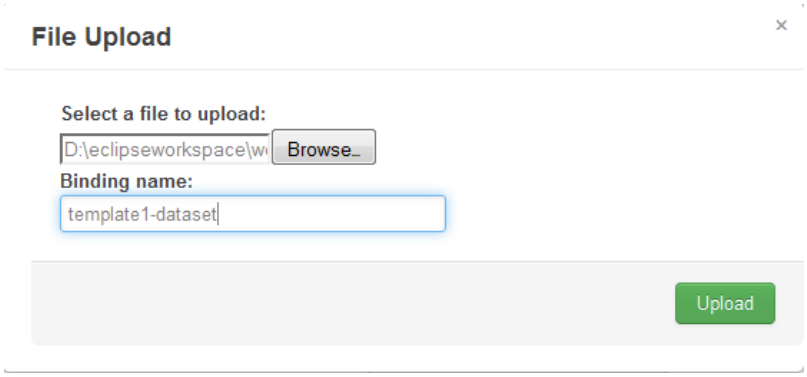


Figure 3-17. Dataset uploading dialogue

Choose the dataset file on your computer with the "Browse" button, and specify the corresponding binding name for it. The binding name tells the system which template it should use with the new dataset

to regenerate the content. It should be in the format of "<*templatebindingname*>-dataset". To find out the binding name of the template you want, you can use the resource view tools that will be introduced in Section 3.3. After uploading your dataset, the eBook will then regenerate and update all the related dynamic content automatically.

3.2.6 Table of contents

A table of contents is located to the left of the reading interface. It is rendered in a tree view. The current section you are reading is highlighted in bold and this is automatically updated as you navigate the content in the reading area by scrolling or changing page. You can also use it to navigate around the content of the eBook, simply clicking on a section name to go to the beginning of that section.

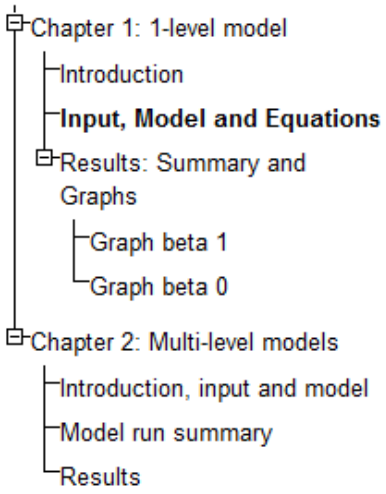


Figure 3-18. The table of contents view

By default, the tree view of the table of contents is fully extended, but you can click on the '+' or '-' sign, to the left of the corresponding higher-level section name, to extend or collapse its sub-content.

As mentioned earlier, eBooks usually contain dynamic content, and this can be inserted or replaced during your reading, with the table of contents automatically updating itself to reflect any such changes. For example, if we compare Figure 3-14 and 3-15 you will notice that 'Graph beta 1' and 'Graph beta 0' have appeared in the table of contents in the latter figure as the result of the user submitting their parameter selection, even though the reading process has not reached that particular position in the eBook yet.

3.3 eBook Resource View

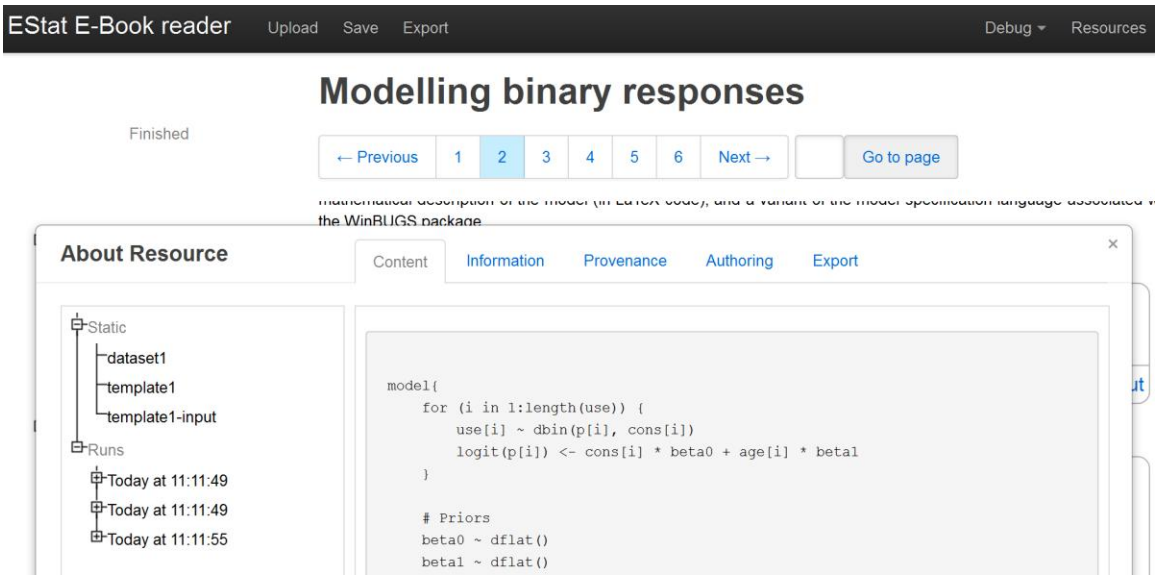


Figure 3-19. The resource view

Some readers may be interested in information beyond that shown in the reading area, and the eBook resource view has therefore been developed to provide further details about the executions behind the content. The resource view contains the structure of the resources used in the current part of the eBook, and reveals more information about each resource by allowing the reader to see the content, meta-information, and provenance of it. It also allows the reader to export certain resources to be reused in other applications. Figure 3-18 shows the resource view interface; this consists of a large dialogue box with a resource tree view on the left-hand side, and an information area which displays further information about a particular resource in five tabbed panels.

There are two ways to access the resource view: either via the ‘Resource’ button in the system menu (the black bar at the top), or by clicking the ‘About’ button on the lower right-hand side of any dynamic output in the eBook reading area. If you open the resource view by clicking an ‘About’ button, the resource view will be opened with the information relating to that dynamic resource displayed. If you instead access it from the system menu, the information area of the resource view will initially be blank, and you will need to click on a resource in the resource tree view in order to view its information.

3.3.1 Resource Tree view

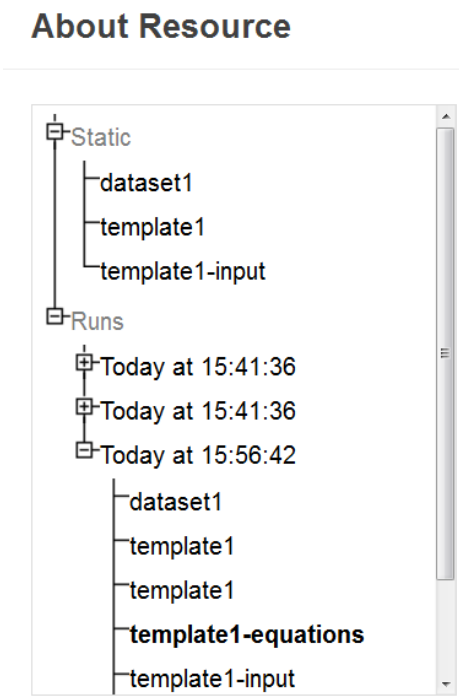


Figure 3-20. The resource tree

The resource tree view shows the resources used in the current section of the eBook divided into two categories: 'Static' and 'Runs'. The 'static' resources are usually those included by the eBook author to enable the executions; these may include execution templates, datasets and predefined input sets. Under the 'Runs' category you will find the executions that have been carried out in the current eBook section. These are displayed and ordered by their start time. Each of the executions can be further extended to show a sub-tree of all the resources used and generated by it. This allows the reader to access the resources that the author may not have shown in the eBook reading content, such as the actual template and dataset used, and many execution outputs that are generated but not used in the eBook reading content by the author. The resource which has its information currently displayed in the resource view is highlighted in the resource tree view.

3.3.2 The Information area

There are five tabs in the information area. The content tab displays the actual content of the resource, whilst the information tab shows some general descriptive information concerning it. The provenance tab shows the provenance of a resource: this is the information generated during the reading process regarding the resource's relationship with the reading process, execution processes and other resources. The export tab provide the link for the user to export the resource in certain format available. Currently it supports exporting the content of resources in CSV format. More detail can be found in Section 3.3.4. The authoring tab provides useful tools for eBook authors, please refer to Chapter 4 for more details of this panel.

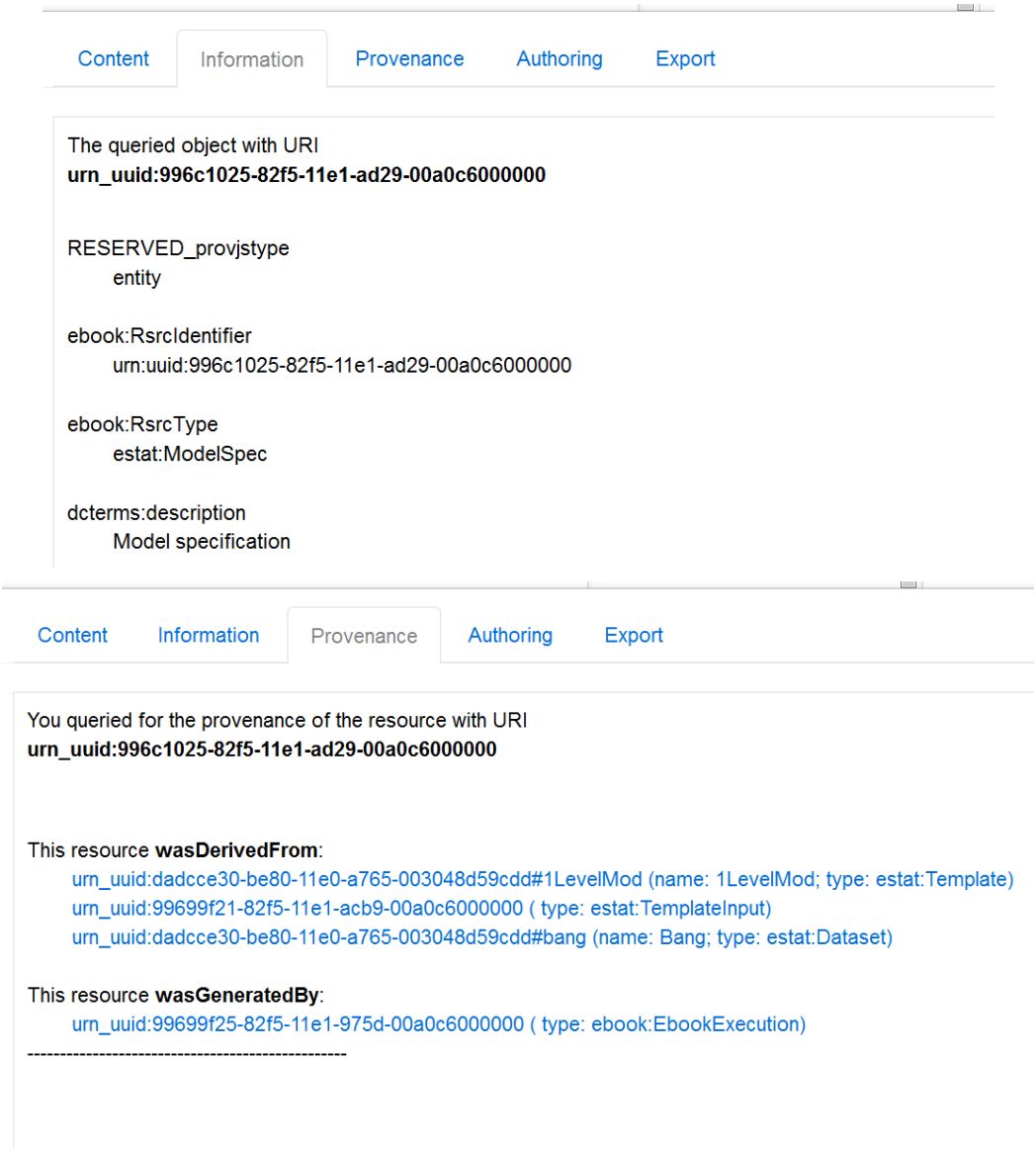


Figure 3-21. The information and provenance panel

3.3.3 Navigation in the Resource View

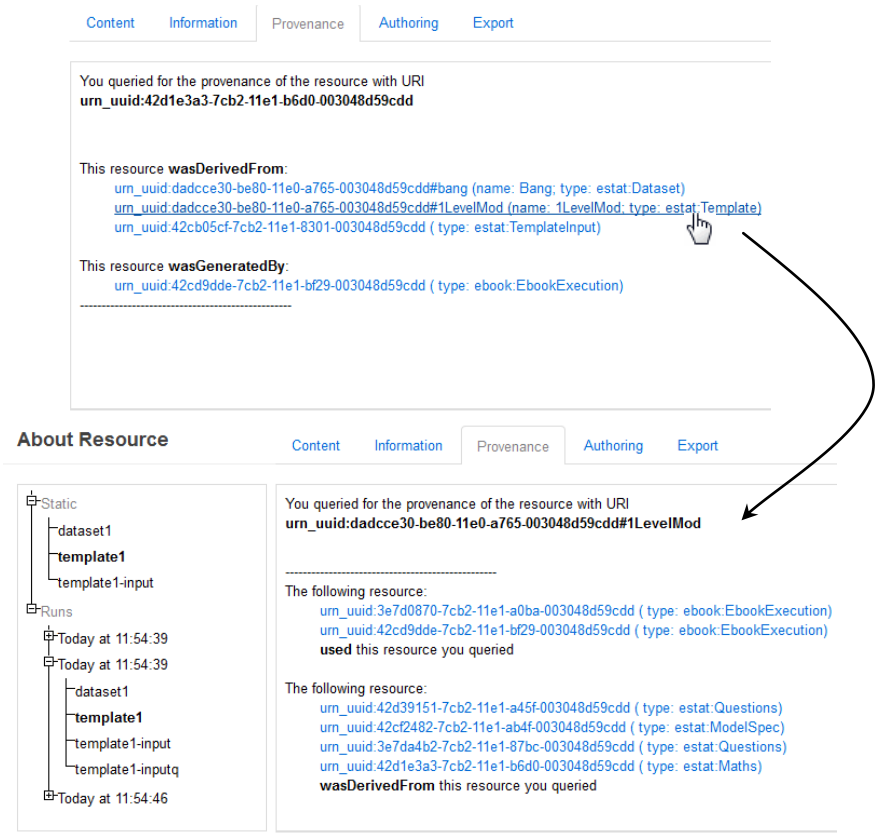


Figure 3-22. Navigate within the resource view

You can navigate around different resources by clicking on them in the resource tree; the information in all tabbed panels will then be updated as you do so. Alternatively, you can click on a resource in the information area if it is available as a link. The information in all tabbed panels will then be updated and the resource you clicked on will be highlighted in the resource tree view.

3.3.4 Exporting a resource

You can export the resource you are viewing through the link in the "Export" tab shown in Figure 3-22. After you click on the link your browser will ask you to save the export file in a popup window. In the example shown in Figure 3-23 and 3-24 the resource exported is a "data output" resource that is exported as a CSV file. Currently this is the only resource type you can export. More resource types will be supported in future updates.

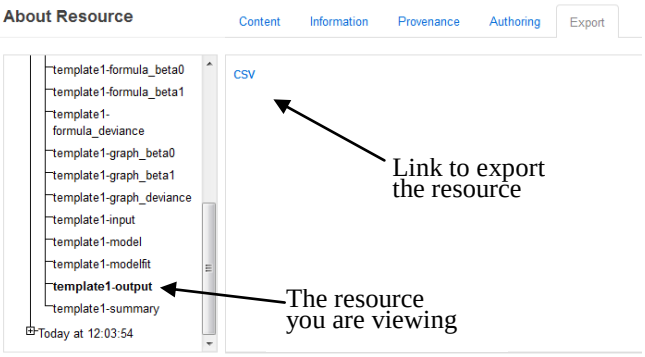


Figure 3-23. Exporting a resource

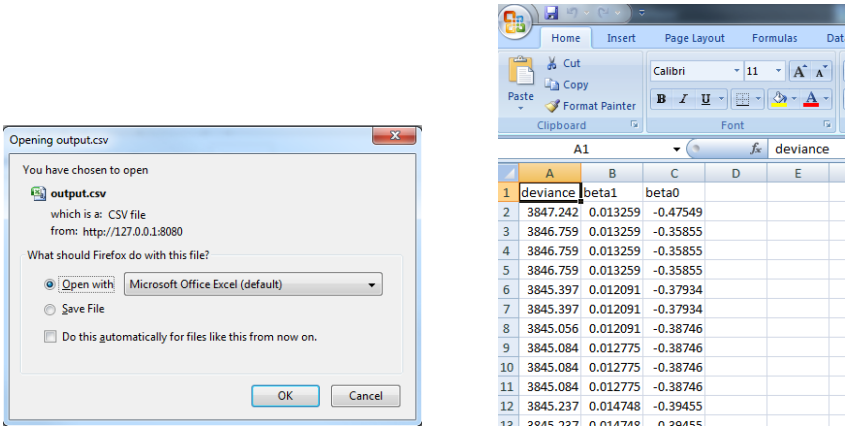
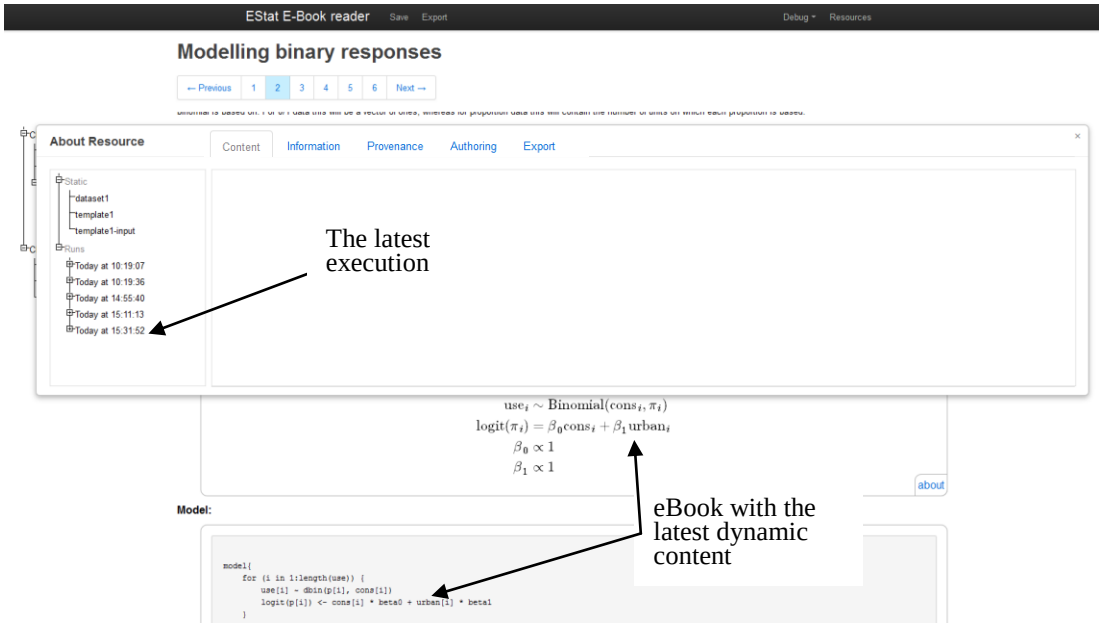


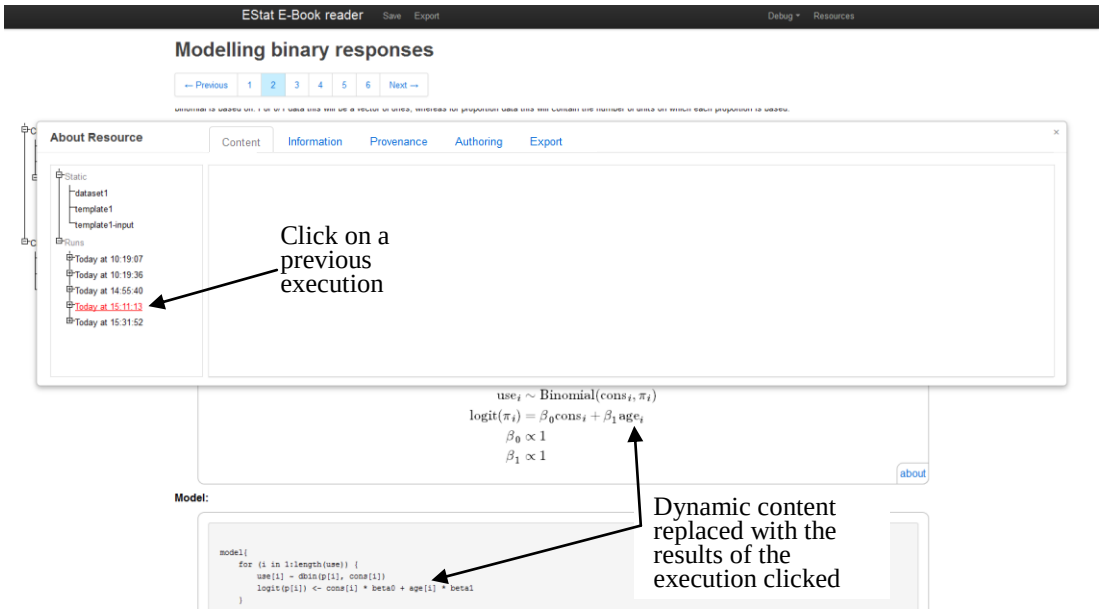
Figure 3-24. Save and open the export file

3.3.5 Reversing the eBook content to a previous state

In Section 3.2 we learnt how to regenerate the dynamic content multiple times by giving different answers to an input question. The eBook content will always be updated with the latest dynamic content based on your last answer. However, with the resource tree view, you can reverse the content in the current section of the eBook to a previous state that was displayed at the end of a specific execution in your reading history.



(a)



(b)

Figure 3-25. Reversing eBook content to a previous state

To do this, simply click, in the resource tree view, on the execution you want to reverse to, and the dynamic content in the current section will be replaced with that generated by this execution. Figure 3-25 shows an example of state-reversing. In a prior execution we selected 'cons' and 'age' as inputs, which triggered an execution labelled "Today at 15:11:13". After reading the results of our initial selections, we then selected 'urban' instead of 'age', and this triggered the last execution which generated the dynamic content displayed in Figure 3-25(a), labelled "Today 15:31:52". To reverse the eBook state we simply

click on the execution "Today at 15:11:13" in the resource tree. As shown in Figure 3-25(b), the eBook replaces the dynamic content with the results of the earlier execution.

4 ADVANCED GUIDE FOR EBOOK AUTHORS

4 ADVANCED GUIDE FOR EBOOK AUTHORS

This chapter describes how to author an eBook, and is organised as follows: Section 4.1 reveals the structure inside an eBook file and how the resources contained in an eBook are organised and bundled together; Section 4.2 describes the types and formats of the resources that can be contained in an eBook; finally, Sections 4.3 and 4.4 provide a step-by-step guide to authoring an eBook by describing the two key files: the 'ebookdef.ttl' file that defines the resource organisation, and the 'ebookpages.html' file that contains the content of the eBook together with its format.

4.1 The eBook - an inside view

4.1.1 File Structure

An importable eBook is in the form of a zip file that contains all its content and resources. The file structure inside the zip file is illustrated by the example shown in Figure 4-1.

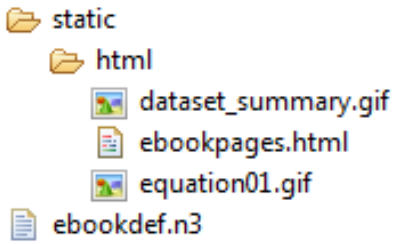


Figure 4-1. File structure inside an eBook file

Currently there are two main parts:

- the static eBook content in the folder '/static/html/';
- the resource organisation definition file 'ebookdef.ttl'.

4.1.2 Organization of Resources

Understanding the logical organisation of resources contained in an eBook is essential to making an eBook execute correctly.

The basic resources contained in an eBook include:

- the templates: the self-contained objects which perform specific tasks. These are written by advanced users and are usually in the form of a python module file;
- the datasets: the raw data sets consumed by the execution templates, usually in the form of a STATA data file with a ".dta" extension;

- the inputs: the parameter values required by the execution templates to carry out their tasks;
- the eBook content: the actual content that will be shown to the eBook readers, including text, images, tables and template outputs.

The eBook content is further organised into multiple activity regions, which can be thought of as different book chapters. Figure 4-2 shows the logical organisation of these resources.

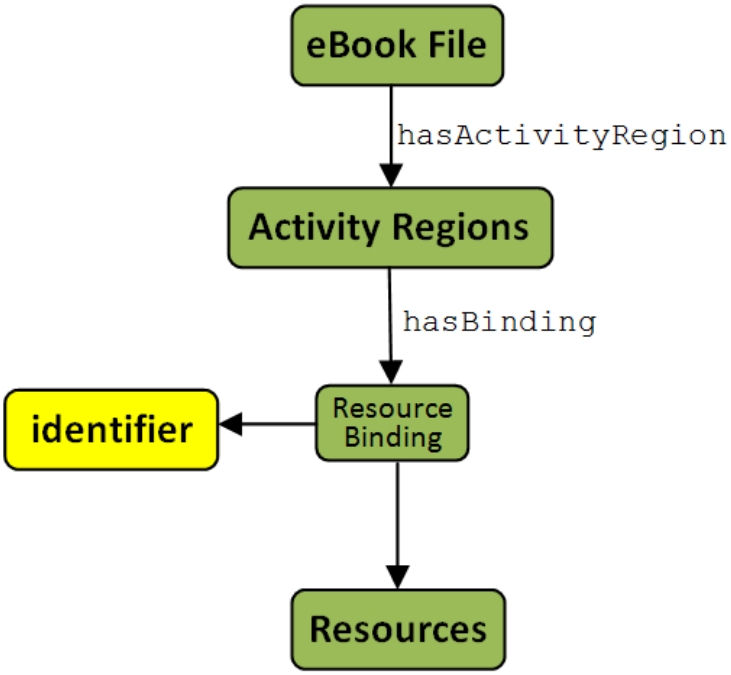


Figure 4-2. Logical Organisation of eBook resources

The activity regions

There is no limit to how long/large an eBook can be, nor a limit on how many resources can be contained within it. So, in theory, the author could make an eBook that has hundreds of pages and contains as many resources – such as template and dataset files – as he or she likes. However, for most of the time during the eBook reading process, only a small part of the content is shown, while only the resources that are related to that part of content are being used; it is not efficient for the system to always load up all the resources contained in an eBook while only using a small selection of them at any given time. As a result, the concept of an 'activity region' has been developed to separate the eBook content into multiple sections. Each resource, such as templates, datasets and input sets, are then associated with the activity region in which they are used.

Resource bindings

Rather than linking a resource directly with an activity region, though, we implement the association through a binding resource. This allows for a particular resource to be used in several different activity regions, but with different binding names. A binding resource represents a ternary relationship that attaches a unique binding name that is given by the author to the resource-activityregion relation. It is bound exclusively to one of the activity regions and can be uniquely identified by its identifier. This avoids confusion and the need to make multiple copies of a given resource.

Example

Here we describe an example eBook to illustrate the concept of an activity region and resource binding. This example eBook will also be used in the other sections later in this chapter.

For this example we have created an eBook that describes modelling binary data with Stat-JR. The structure of the example eBook is shown in Figure 4-3. The eBook explains how to fit single-level and multi-level models, highlighting the difference between them, by fitting the model using two different templates, '1LevelMod' and '2LevelCat'. The same dataset, 'Bang', is used throughout. We divide the eBook into two activity regions – ActivityRegion001 and ActivityRegion002 – that contain the resources for the two templates, respectively. The author can use any combination of letters and numbers for the activity region identifiers as long as they are unique in this eBook.

We then create the binding resources. Each activity region will need to be bound to the templates that the author would like to be used within it, as well as those templates' input, and also a dataset. For each binding resource, an identifier is given by the eBook author as an internal name. Again, the author can use any combination of letters and numbers for the identifier as long as it is unique in this eBook. The binding resources for the dataset in both activity regions point to the same dataset resource, but with different identifiers.

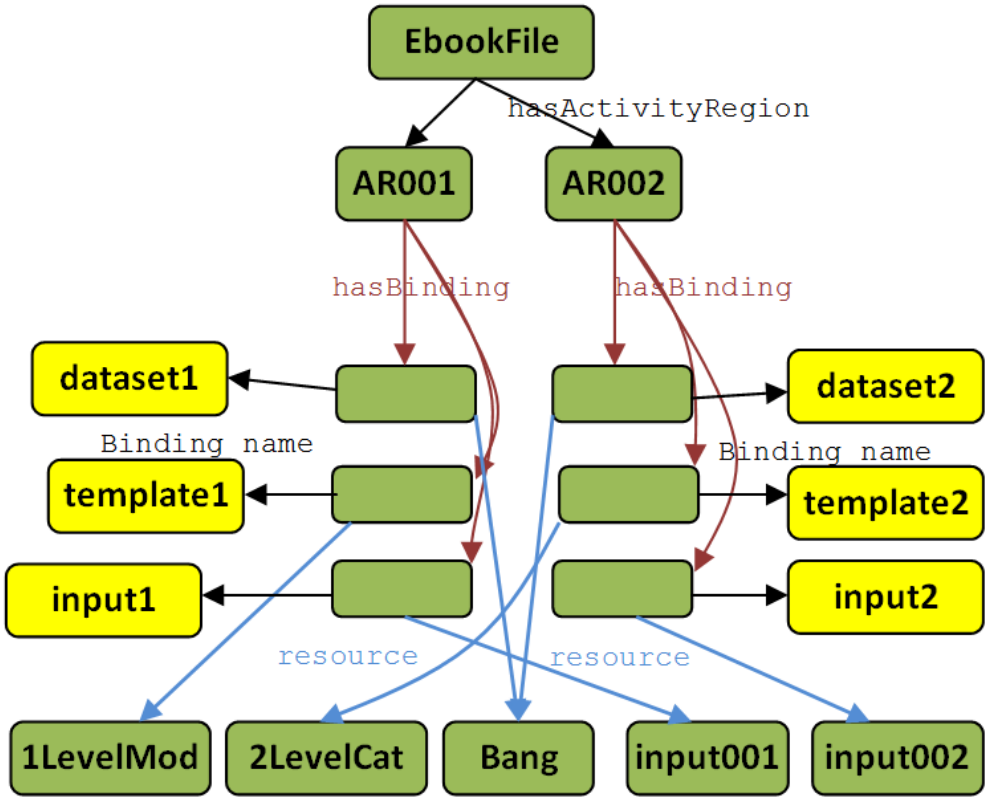


Figure 4-3. Example of the organisation of eBook resources

4.2 Preparing the Resources

4.2.1 Template and Dataset File

The template files are written by users who are usually specialists in building algorithms in certain fields and disseminating them as new templates. The STAT-JR software also provided many templates, including the two used in the previous example, that have been developed by the team of programmers behind STAT-JR. The dataset files can easily be assembled by any user in the form of a STATA '.dta' file. This User Guide focuses on the resources that are used exclusively in the eBook system. For details of how to produce templates and .dta files please refer to the corresponding documentation concerning Stat-JR.

4.2.2 Template Input File

The inputs for a template can be given either in the ebookdef.ttl file or in a separate input file. If you want to define an input set as part of the 'ebookdef.ttl' file, please refer to Section 4.3. In the case of a separate input file, the eBook system can read inputs in two formats: Python and JSON.

Input file as a Python module

To provide the inputs in a python module, write your input set in the form of a Python dictionary, and save it as a *.py* file. A Python dictionary is a container type that consists of pairs (called items) of keys and their corresponding values. Each key is separated from its value by a colon (:), the items are separated by commas (,), and the whole container is enclosed in curly braces({}). To write your input set as a Python dictionary, put each input name as an item key and the corresponding input value as the associated item value. Both the input name and value should be given as a Python string, which means they must be wrapped by either single quotes (') or double quotes ("). An example of an input set in Python dictionary format is as follows:

```
{"y" : "normexam",
 "Engine" : "OpenBUGS",
 "EstM" : "yes",
 "outdata" : "out",
 "seed" : "1",
 "burnin" : "1000",
 "iterations" : "1250"}
```

Input file as JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format that is designed to be easy for people to read and write. It is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages. In practice, an input set in the form of a JSON collection of key-value pairs looks exactly the same as a Python dictionary except it is saved as a *.json* file.

4.2.3 Resource File Location

When you use a template or a dataset that comes with the STAT-JR system, you do not need to worry about its location as it's already in the default directory of the STAT-JR system. For more information on the default templates and datasets, please refer to the Stat-JR documentation.

To provide your own template, dataset or input files, save the files in the top level directory of your eBook. If you are starting from an eBook authoring template, the top level directory is the level where the ebookdef.ttl file is located.

4.3 ebookdef.ttl – Defining the Resource Organisation

The author should provide the following information in the ‘ebookdef.ttl’ file:

- General information concerning the eBook file
- Activity regions
- Resources for eBook execution – templates, datasets and input sets
- Binding resources

We will use the example eBook described in Section 4.1 to explain how to make an eBook definition file.

The Turtle format

The resource organisation of an eBook is defined in the format of RDF N3 in the 'ebookdef.ttl' file. The Turtle, is a serialisation of Resource Description Framework models, is much more compact and readable than XML RDF notation. More information about the N3 format can be found at http://www.w3.org/TR/turtle/.

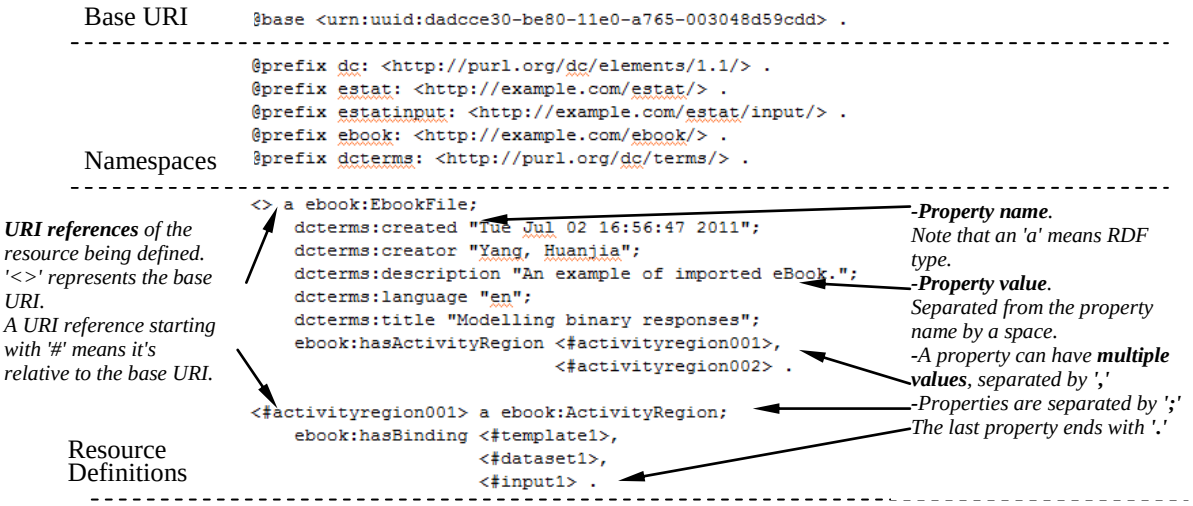


Figure 4-4. Example of an eBook definition file in Turtle format

Figure 4-4 shows the beginning of an example 'ebookdef.ttl' file, which contains all the Turtle syntax we use in the eBook system. There are three parts to the eBook Turtle file: the base URI, which is a string of characters used to uniquely identify the eBook over the Web, the namespaces and the resource definitions.

Base URI - The eBook file identifier

The eBook file needs to be identified uniquely – this is achieved via an URI defined using the Universal Unique IDentifier (UUID). The author can use any tool that generates a standard UUID identifier (including the 'UUIDURI' eBook provided by default in the package), and use it as the base URI at the beginning of the RDF Turtle file.

```
@base <urn:uuid:dadcce30-be80-11e0-a765-003048d59cdd> .
```

Note the full stop sign '.' required by the Turtle syntax at the end of the line.

Namespace definition

The namespaces used must be defined at the beginning of the file. The namespaces used in the eBook definition are as follows:

```
@prefix estat: <http://purl.org/net/statjr/ns#> .
@prefix estatinput: <http://purl.org/net/statjr/input#> .
@prefix ebook: <http://purl.org/net/deep/ns#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
```

Simply copy this portion of code and put it at the beginning of the Turtle file after the base URI definition. Also note the full stop sign '.' at the end of each line.

Providing general information about the eBook file

The author should also provide some general information about the eBook here. There are four mandatory properties: `dcterms:created`, `dcterms:creator`, `dcterms:description` and `dcterms:title`; these provide the creation date, author name, short description, and the title of the eBook, respectively. The definition of the example eBook described in Section 4.1 is shown below.

```
<> a ebook:EbookFile;
   dcterms:created "Tue Jul 02 16:56:47 2011";
   dcterms:creator "Yang, Huanjia";
   dcterms:description "An example of imported eBook.";
   dcterms:language "en";
   dcterms:title "Modelling binary responses".
```

Please use a space to separate the name and value of each property. The value of each property should be wrapped by double quote signs. Use a full stop sign at the end of the line of the last property, and a semicolon for the others.

Declaring the activity regions

The activity regions should be declared with the property `ebook:hasActivityRegion`. Each activity region should be given a unique name in the eBook. As before, the author can use any combination of letters and numbers for the identifier as long as it is unique in this eBook. The activity region declarations of the example eBook described in Section 4.1 are shown below.

```
<> a ebook:EbookFile;
   dcterms:created "Tue Jul 02 16:56:47 2011";
   dcterms:creator "Yang, Huanjia";
   dcterms:description "An example of imported eBook.";
   dcterms:language "en";
   dcterms:title "Modelling binary responses";
   ebook:hasActivityRegion <#activityregion001>,
                           <#activityregion002> .
```

```
<#activityregion001> a ebook:ActivityRegion .

<#activityregion002> a ebook:ActivityRegion .
```

Note a '#' sign is put in front of the activity region names to turn them into URIs under the base URI defined at the beginning of this section. The name is then declared as an activity region by defining its RDF type as an `ebook:ActivityRegion.`

Declaring the eBook execution resources

The resources used in the eBook need to be declared. These include the templates, datasets and input sets. The resource declarations of the example eBook described in Section 4.1 are shown below.

```
<#1levelMod> a estat:Template;
             ebook:hasName "1LevelMod";
             estat:isBuiltIn "1LevelMod" .

<#2levelCat> a estat:Template;
             ebook:hasName "2LevelCat";
             estat:isBuiltIn "2LevelCat" .

<#bang> a estat:Dataset;
        ebook:hasName "Bang";
        ebook:hasFilename "static/bang.py" .

<#input001> a estat:TemplateInput;
        ebook:hasName "Input001";
        ebook:hasFilename "input001.py" .

<#input002> a estat:TemplateInput;
             estatinput:L2ID "district";
             estatinput:D "Binomial";
             estatinput:link "logit";
             …… .
```

Again, note that a '#' is put in front of the resource names to turn them into URIs under the base URI defined at the beginning of this section. For each of these resources, the author needs to give a name with attribute ebook:hasName and point it to the file name of the physical resource with attribute ebook:hasFilename or estat:isBuiltin. The difference is that ebook:hasFilename is used to specify the name of the physical file saved in the disk or attached with the eBook and estat:isBuiltIn is used to refer to a resource that comes with the StatJR system by default. As the file name does not always specify what is contained, the ebook:hasName is there for author to specify a nick name for the file. This name is not used by the system to drive any functionality, except being displayed as an attribute of the file resource to the user in the information tab of resource view. The type of each resource is defined by the value of its RDF type property, which is presented as "a" in the Turtle file. The type value consists of a name space, which indicates whether the resource is a e-Stat type or a eBook exclusive type, and a type name. Table 4-1 lists the RDF types used in the ebookdef.ttl file.

Type of resource	RDF name space used	RDF type name
eBook file	ebook	EbookFile
Activity region	ebook	ActivityRegion
Binding resource	ebook	ResourceBinding
Execution template	estat	Template
Dataset	estat	Dataset
Input set	estat	TemplateInput

Table 4-1. RDF types used in the ebookdef.ttl file

Please note that the type names are case-sensitive. Currently the input value pairs are defined directly in the ebookdef.ttl file as properties of the input sets.

Defining binding resources

The declared eBook execution resources then need to be associated with corresponding activity regions. For each resource association, a binding resource needs to be created with a unique binding identifier, as well as an `ebook:resource` property pointing it to one of the resources already declared. The activity regions are then linked to the appropriate binding resources by putting the URI of the binding resources in the value of property `ebook:hasBinding`. The value of binding resource property `ebook:identifier` is the unique identifier of that binding resource in the eBook. It will be later used in Section 4.4 as the prefix of the dynamic output names in the eBook content. The binding resource declarations of the example eBook described in Section 4.1 are shown below.

```
<#activityregion001> a ebook:ActivityRegion;
                     ebook:hasBinding <#template1>,
                                      <#dataset1>,
                                      <#input1> .

<#activityregion002> a ebook:ActivityRegion;
                     ebook:hasBinding <#template2>,
                                       <#dataset1>,
                                       <#input2> .

<#template1> a ebook:ResourceBinding;
             ebook:identifier "template1";
             ebook:resource <#1levelMod> .

<#template2> a ebook:ResourceBinding;
             ebook:identifier "template2";
             ebook:resource <#2levelCat> .

<#dataset1> a ebook:ResourceBinding;
            ebook:identifier "dataset1";
            ebook:resource <#bang> .
```

```
<#input1> a ebook:ResourceBinding;
          ebook:identifier "template1-input";
          ebook:resource <#input001> .

<#input2> a ebook:ResourceBinding;
          ebook:identifier "template2-input";
          ebook:resource <#input002> .
```

Note:

- If a property value is a URI, it should be wrapped by angle brackets rather than double quotes. When there are multiple values for a property, use commas to separate the values.
- You can bind multiple templates with an activity region. They will be executed in a queue when the eBook reader enters the activity region. However, currently only one dataset can be bound in each activity region to be used by the templates contained. The system does allow the user to upload their own dataset during reading to perform executions with those templates, please refer to Section 3.2.5 for more details.

4.4 ebookpages.html – Writing the Content

After providing general information in the Turtle file, the execution resources and their organisation are then defined. The author can now edit the content (i.e. the information the eBook readers will be able to see) in the 'ebookpages.html' file.

In the eBook system, the content is written in standard HTML. Although there are extensive choices of HTML editor available, some basic knowledge in reading and writing raw HTML would be helpful for eBook authors to organise an eBook quickly and correctly. The system defines several HTML markers that the author can use to make pages and activity regions, to specify when content should be hidden or shown, and to mark the positions at which specific dynamic content will be inserted. The author does not need to add any HTML headers or styles: the eBook system will add these automatically before rendering the content to the eBook readers.

The 'ebookpages.html' file of the example eBook described in Section 4.1 can be found in Appendix II.

4.4.1 Adding Static Content - Text, Images and LaTeX

Static text can be put directly into the ebookpages.html file in a similar manner to adding content in a HTML web page. The eBook system will do general styling for this content. It also supports any HTML inline style that you include. For example, you can highlight a word, or a part of text, by making it bold or coloured.

Static images are also inserted in the standard HTML manner. Simply put the image file in the same folder as the ebookpages.html file, and then in the file content add the following marker in the position you want the image to be displayed:

```
<img src="image_file_name"/>
```

In order for the system to style it correctly, we suggest that you wrap the image within an HTML block with the pre-styled class "*deep_static_img*". An example, with an image name, follows:

```
<div data-deep-id="dataset_summary01" class="deep_static_img">
    <img src="dataset_summary.gif" alt="" /><br>
    Figure 1
</div>
```

Tables can also be inserted in the form of standard HTML tags.

You can also insert mathematical symbols and equations in LaTeX code. For the LaTeX code to be identified and rendered by the system you need to wrap it with '\(' and '\)', which makes an inline maths equation, or '\[' and '\]', which produces the equation in the middle of a new line. For example, the following html code:

```
<p>This is an equation: \( \mbox{normexam}_i \sim \mbox{N}(\mu_i, \sigma^2) \).
It is an example.</p>
```

produces:

This is an equation: $\mathrm{normexam}_i \sim \mathrm{N}(\mu_i, \sigma^2)$. **It is an example.**

While the following code:

```
<p>This is an equation: \[ \mbox{normexam}_i \sim \mbox{N}(\mu_i, \sigma^2) \]
It is an example.</p>
```

produces:

This is an equation:

$$\mathrm{normexam}_i \sim \mathrm{N}(\mu_i, \sigma^2)$$

It is an example.

4.4.2 Making Chapters and Pages

The content needs to be divided into activity regions as defined previously in the N3 file. The author needs to use the HTML div syntax with class *"deep_activityregion"* to wrap the content in each activity region. If the content in an activity region is long, then it can be divided into different pages. Please note that pages must be fully contained in one activity region, and they cannot nest other activity regions or pages. To do this simply wrap the content of each page into HTML divs with class

"deep_page" as the following example illustrates; the eBook system will then render your content page-by-page during a reading process.

```
<div id ="activityregion001" class="deep_activityregion">
     <div id ="ebookpage001" class="deep_page">
         Content of page001
         <div class="deep_static_img">
             <img src="example_static_image.gif" alt="" /><br>
             Figure 1
         </div>
     </div>
      <div data-deep-id="ebookpage002" class="deep_page">
         Content of page002
      </div>
      <div data-deep-id="ebookpage003" class="deep_page">
         Content of page003
      </div>
</div>

<div id ="activityregion002" class="deep_activityregion">
     <div id ="ebookpage004" class="deep_page">
         Content of page004
     </div>
     <div data-deep-id="ebookpage005" class="deep_page">
         Content of page005
     </div>
     <div data-deep-id="ebookpage006" class="deep_page">
         Content of page006
     </div>
</div>
```

The names of the activity regions need to match the names you have defined in the N3 file, but without the symbol '#'. The name of each page can be any combination of letters and numbers for the identifier, as long as it is unique in this eBook.

Please note that the activity region wrapping is the top-level marker of your content; they cannot nest within each other. Any content that is not wrapped in an activity region div will NOT be rendered to the eBook reader.

4.4.3 Chapter and Section Headers

You can add your chapter and section headers using standard HTML header markers. You are free to use all available header levels (h1-h6) in HTML to structure the chapter and section headers into a proper hierarchy; this will then be rendered in the ‘content outline’ section during the reading process. You need to add the chapter and section numbers together with the header name if required, as the eBook system will not number them.

If you are making one activity region for each chapter, you can allow the chapter header to be displayed at the top of all the chapter pages by putting the header outside page divs at the beginning of the activity region. The following illustrates the content of the example eBook described in Section 4.1, with each activity region containing one (and only one) chapter. For means of illustration, the headers added are highlighted here by a grey background.

```
<div id ="activityregion001" class="deep_activityregion">
<h1>Chapter 1: 1-level model</h1><hr>
     <div id ="ebookpage001" class="deep_page">
     <h2>Introduction</h2>
         Content of page001
         <div class="deep_static_img">
             <img src="example_static_image.gif" alt="" /><br>
             Figure 1
         </div>
     </div>
     <div data-deep-id="ebookpage002" class="deep_page">
      <h2>Input, Model and Equations</h2>
         Content of page002
     </div>
     <div data-deep-id="ebookpage003" class="deep_page">
     <h2>Results: Summary and Graphs</h2>
         Content of page003
     </div>
</div>

<div id ="activityregion002" class="deep_activityregion">
<h1>Chapter 2: Multi-level models</h1>
     <div id ="ebookpage004" class="deep_page">
      <h2>Introduction, input and model</h2>
         Content of page004
     </div>
     <div data-deep-id="ebookpage005" class="deep_page">
     <h2>Model run summary</h2>
         Content of page005
     </div>
     <div data-deep-id="ebookpage006" class="deep_page">
     <h2>Results</h2>
         Content of page006
     </div>
</div>
```

4.4.4 Adding Dynamic Content

At the point in the document where you want to include some dynamic content, simply add a HTML div with class *"deep_dynamic_output"* and give the div id in the format of <*template_bindingname*>-<*output_name*>. For example, if you want to show the output 'equations' of the execution of 'template1', add the following div in the position you want it to appear:

```
    <div id ="template1-equations" class="deep_dynamic_output">
    </div>
```

Please note that the <*template_bindingname*> is the identifier of the binding resource that links the template with the activity region. It is not the name of the actual template resource. So in this case it is `id ="template1-equations"` rather than `"1LevelMod-equations"`. The content of the div will be replaced with the dynamic output after the execution engine generates it.

The <output_name> of the dynamic content varies for different execution engines, and also for the different templates you used. Please refer to the documentation of each execution template you want to use to find out the dynamic outputs it can generate, and the name to be used in the eBook file. The eBook system itself does not have the knowledge of what output will be generated, it simply take what the execution engine throws out, and match those with the dynamic content names you put in the eBook content.

Adding a template parameter input area

Instead of giving a value for all template input parameters, you can leave some of them to be defined by the reader during their reading. This enables the reader to try out different parameter values for the same template execution, and then compare results.

The eBook system uses the dynamic output mechanism described in Section 4.4.4 to acquire inputs from the reader. When an eBook execution triggered behind the scenes does not have all the parameter inputs for the template it uses, the execution generates a dynamic output with type 'inputq', and terminates. The actual content of the 'inputq' output is basically some HTML code rendered to the reader as an input question. The input question could be in the form of a text box, a select box or a drop down box, and this depends on the type of input parameter you have asked the reader to enter. The content of the 'inputq' output also contains a 'submit' button that triggers a new execution after the user enters the parameter value required.

To let the reader enter some of the input parameters for a specific template in the eBook, simply remove the line in the 'ebookdef.n3' file or the input file that defines that parameter and add a dynamic output area with id <*template_bindingname*>*-inputq* in the 'ebookpages.html' file. For example, to let the reader enter the value of parameter 'x' for 'template1' during reading, remove the line that defines the value for input 'x' in the 'ebookdef.n3' file or input file, and add the following dynamic output div at the position where you want the input question to appear:

```
<div id ="template1-inputq" class="deep_dynamic_output">
</div>
```

As the input question is handled as dynamic output in the eBook, you can also provide default content or add static content to be displayed with it using the methods described in the previous section.

Adding a dataset summary

You can add a dynamic output that displays the summary of a dataset in the current activity region. The dataset summary will be shown before the template(s) in the same activity region being executed. To add such an output, put a div with class *"deep_dynamic_output"* and id in the format of *<dataset_bindingname>-summary* at the position where you want the data summary to appear. For example, to show the summary of a dataset with binding name dataset1, add the following code:

```
<div data-deep-id="dataset1-summary"
     class="deep_dynamic_output deep_dynamic_hidden">
</div>
```

As the dataset summary is handled as dynamic output in the eBook, you can also provide default content or add static content to be displayed with it using the approaches described in the previous section.

Default content for dynamic output

You can put static content inside the dynamic output div either as a default display, or as a message to the reader before the output is generated. For example, the following div means that the eBook will display a line of text reading 'The summary will be displayed here after the model fitting' in the position of the summary of template1 before it is actually generated.

```
<div id ="template1-summary" class="deep_dynamic_output">
    The summary will be displayed here after the model fitting.
</div>
```

You can put in any type of static content, but please note that after the corresponding dynamic output is generated, it will replace all the static content inside the div, so do not include any content that should always be shown inside the dynamic output divs.

4.4.5 Accessing and configuring dynamic output

Accessing the data in dynamic output

Name	Mean	Std	Count
woman	1434.00000	827.63156	2867
district	29.25323	17.86300	2867
use	0.39728	0.48933	2867
use4	3.19847	1.08548	2867
lc	1.66864	1.24069	2867

Figure 4-5. Example of a dataset summary output

The eBook system renders the output in a simple XML format. For example, for a dataset-summary output in Figure 4-5, the XML format looks like:

```
<tabular> <heading>
          <element>Name</element>
          <element>Mean</element>
          <element>Std</element>
          <element>Count</element>
          </heading>
          <rows>
          <row row="woman">
          <element col="Mean">1434.00000</element>
          <element col="Std">827.63156</element>
          <element col="Count">2867</element>
          </row>
          ...
</tabular>
```

Then in your eBook content, using an attribute `data-deep-expression` you can define the path to the value you want to access and include it in your document. The expression uses syntax of XPath, which is a language recommended by W3C to navigate through elements and attributes in an XML document. For example, in the following component in an eBook:

```
This dataset has
<span class="deep_dynamic_output" data-deep-id="dataset1-summary"

data-deep-expression="//row[@row='woman']/element[@col='Count']/text()">
</span>
rows.
```

the value of `data-deep-expression` is a XPath expression that indicates the text of the element with attribute name 'Count' in the row with attribute 'row' equals to 'woman'. In this example, when the dataset1-summary output becomes available, this component will be shown in the eBook reader as: *"This dataset has 2867 rows. "*

Selecting the rows and columns to be displayed

The eBook system provides two attributes, `data-deep-only` and `data-deep-except`, for the author to specify which rows and columns of an dynamic output to display. For example, inserting the previous example's dataset1-summary with the following code:

```
<div data-deep-id="dataset1-summary" class="deep_dynamic_output"
data-deep-only="Mean,Count" data-deep-except="cons">
```

will produce the display shown in Figure 4-6, in which the 'Std' column and the 'district' row are not shown.

Name	Mean	Count
woman	1434.00000	2867
use	0.39728	2867
use4	3.19847	2867
lc	1.66864	2867
age	-0.32787	2867

Figure 4-6. Selecting rows and columns to be shown in an tabular output

4.4.6 Show/Hide static element with dynamic content

You can show or hide some static content when a specific dynamic content becomes available in the same activity region. In the previous section we have introduced the way of using the `deep_dynamic_hidden` class to show or to remove some default content when a dynamic content becomes available. However, you can do only one of them (either show or hide), and the content you are showing/hiding has to be in an HTML element that nests the dynamic content. If when a dynamic content becomes available, you would like to hide some parts of content and show some other parts at the same time, or, if the content associated with the dynamic content is not adjacent to it, then you will need to use the `showon/hideon` mechanism in this section. Those two attributes are also useful if you want to show/hide an dynamic content only when another dynamic content becomes available.

Show element when a dynamic content becomes available

If a hidden element is not a parent of a dynamic content, but you want to show the element when the dynamic content becomes available, simply add the element with class `"deep_dynamic_hidden"` so that it is not visible at the beginning. Then add the property data-deep-showon=`"<dynamic_content_name>"` to specify the dynamic content that the element should appear with. For example:

```
<div class="deep_dynamic_hidden" data-deep-showon="template1-inputq">
    This text will appear when template1-inputq becomes available.
</div>
<div id ="template1-inputq" class="deep_dynamic_output deep_dynamic_hidden">
</div>
```

Please note that if you put headers in the static content in the element, they will also be initially hidden in the table of contents block, and will appear in the table of contents only when the corresponding dynamic content becomes available.

Hide element when a dynamic content becomes available

To hide a visible element when a dynamic content becomes available, simply add the element with class *"deep_dynamic_visible"* to make sure it is visible at the beginning. Then add the property data-deep-hideon=*"<dynamic_content_name>"* to specify the dynamic content that the element should disappear on it becoming available. For example:

```
<div class="deep_dynamic_visible" data-deep-hideon="template1-inputq">
    This text will disappear when template1-inputq becomes available.
</div>
<div id ="template1-inputq" class="deep_dynamic_output deep_dynamic_hidden">
</div>
```

Please note that if you put headers in the static content in the element, they will also be removed from the table of contents when the corresponding dynamic content becomes available.

Use the output data in further conditions for Showon/Hideon

You can specify conditions for showon/hideon an element based on the value you accessed. for example, in the following component in an eBook:

```
<p class="deep_dynamic_hidden" data-deep-showon="dataset1-summary"
   data-deep-expression="//row[@row='woman']/element[@col='Count'
and .>1000]">
Dataset has more than 1000 rows.
</p>
<p class="deep_dynamic_hidden" data-deep-showon="dataset1-summary"
   data-deep-expression="//row[@row='woman']/element[@col='Count'and
not(.>1000)]">
Dataset has no more than 1000 rows.
</p>
```

the XPath expression is as the previous example except it returns the result of a conditional statement instead of the text. This result is used by the eBook system to decide whether to perform the `data-deep-showon` (or a `data-deep-hideon`) action or not. In this example, as the count is 2867, which is larger than 1000, this component will be shown in the eBook reader as: " Dataset has more than 1000 rows. "

4.5 Authoring Tools

4.5.1 eBook Reloading Tool

While authoring an eBook, most authors want to be able to check the result of their latest changes by constantly viewing their eBook draft in the eBook system. However, this is not straight forward with the system's reading interface. When an eBook file is imported, the eBook system decompresses it, puts the content in a folder named by the key of the eBook in the system and puts it in the

"var/ebooks/" directory. Generally, authors create their new eBooks by modifying an authoring template or an existing eBook file. In order to view the eBook with the latest changes, we suggest that the most convenient way is to import an authoring template or an eBook that you would like to modify, and then edit the content directly in the folder in the "var/ebooks/" directory. In this case, for changes made in the ebookpages.html file, you will be able to see the result after re-entering the reading process.

However, the information contained in the ebookdef.ttl file of an eBook is loaded into the eBook system database when the eBook file is imported. After that, the system uses only the information in the database and does not access the ttl file any more. This is to ensure the speed and consistency of the system. But for eBook authors who are editing the ebookdef.ttl file, this mechanism makes it difficult to view the result of their latest changes to the eBook in the system. Each time the n3 file is changed, the author will have to compress the folder into an eBook file, delete the folder in the "var/ebooks/" directory and import the new eBook file. This can be time consuming and frustrating especially when the author is trying to make many small changes to the ttl file or to compare some different ideas/values in it.

In order to address this issue, the system provides an eBook reloading function that allows the author to reload the information in the ebookdef.ttl file of an eBook into the system database during reading. The function can be accessed from the debug menu of the eBook system's reading interface. With this function, after editing the ttl file of the eBook in the "var/ebooks/" folder, the author can simply press "reload eBook" in the menu to view the result of the latest changes.

Note:

- To ensure system consistency, the reloading function will also erase the reading history related to the current reading process. The eBook reading will be redirected to the first page of the eBook after reloading. All dynamic outputs and execution history of the current eBook will be lost.
- There is now an additional step after pressing "reload eBook": The system will check the eBook file that is about to be reloaded and show possible errors and warnings in a pop up box. At this stage, the authors will need to select whether they want to proceed with the reloading. If "Continue Reloading" is selected, the system will reload the eBook file. If "Cancel Reloading" is selected, the system will abandon the reloading process and go back to the previous reading interface. Please refer to Section 4.5.2 for details of the Filechecker and the error/warning messages.

4.5.2 The eBook Filechecker

The eBook system comes with a file checker that checks for possible issues in an eBook file. It checks the consistency in both the .ttl file and the .html file. File checking is automatically triggered when the author imports a new eBook draft or selects to reload an eBook during reading. It can also be called in the command line.

Errors/Warnings that the file checker can identify in .ttl file:

Syntax errors:

An ebookdef.ttl file with any syntax error cannot be parsed by the system. As the system cannot continue with parsing the Turtle file after reaching the first syntax error, it will only indicate the position and type of the first syntax error even if there are many syntax errors in the Turtle file. In such case the author will have to try checking the Turtle file several times with the system to identify and correct all of them. The system cannot proceed to perform structure and relationship checking until all syntax errors are corrected.

Relationship/structure errors/warnings:

After the Turtle file is cleared of any syntax errors, the system should be able to parse the ebookdef.ttl file. The system will then check for structural and relationship related errors or warnings to ensure all the constraints introduced by the eBook system are satisfied. The possible errors/warnings that will be identified by the filechecker are listed as follows:

1. No EbookFile is defined; (resource related: Ebookfile; type: Error)
2. More than one EbookFile is defined; (resource related: Ebookfile; type: Error)
3. No ActivityRegions defined, an eBook needs at least one; (resource related: ActivityRegion; type: Error)
4. An ActivityRegion is defined but not used by EbookFile; (resource related: ActivityRegion/EbookFile; type: Warning)
5. An ActivityRegion is used by Ebookfile but not defined; (resource related: ActivityRegion/EbookFile; type: Error)
6. Missing required attribute(s); (resource related: All; type: Error)
7. Required attribute(s) has(ve) no value; (resource related: All; type: Error)
8. Optional attribute(s) has(ve) no value; (resource related: All; type: Warning)
9. Naming/URI/Identifier conflict; (resource related: All; type: Error)
10. A binding resource points to a non-existing resource; (resource related: Resourcebinding; type: Error)

11. A resource that is not associated with any binding resource; (resource related: Resource; type: Warning)
12. A binding resource not used by any activity region; (resource related: Resourcebinding; type: Warning)
13. A binding resource used by multiple activity regions; (resource related: Resourcebinding/ActivityRegion; type: Warning)
14. Missing or invalid base URI.
15. Unrecognized resource types are suggested as warnings.
16. A binding resource used by an ActivityRegion does not exist.
17. A binding resource used by an ActivityRegion is not with type ResourceBinding;
18. A resource used by a BindingResource is of not valid type (has to be a Template, a Dataset or a TemplateInput)

Note:

- ActivityRegion URI conflicts cannot be identified/suggested
- Attributes of the resource with type TemplateInput are not checked

Checking ebookdef.ttl with ebookpages.html:

After checking the Turtle file, the filechecker will proceed to check the ebookpages.html file, mainly aiming to identify incorrect usages of the resources defined in the .ttl. The filechecker only checks the ebookpages.html file when there is no error found in the ebookdef.ttl file (warnings are allowed).

Syntax errors:

The system will not check the consistency of the html syntax.

relationship/structure errors/warnings:

The possible errors and warnings related to the ebookpages.html file are listed as below:

1. An ActivityRegion defined in Turtle not used in HTML;
2. An ActivityRegion used in HTML not defined in Turtle;
3. A resource binding name used in HTML not defined in Turtle;
4. A resource binding name defined in Turtle not used in HTML;
5. A resource binding name is not used inside the ActivityRegion that binds with it in Turtle;

Integration of filechecker with eBook system

Checking during eBook import:

When the user imports an eBook file, the system loads the filechecker to check the consistency and displays the error/warning messages in a message page. The errors are the issues identified that can cause the eBook system to malfunction, while the warnings are the issues identified as not complying with the eBook general writing guidelines but are not harmful to the system itself. The background colour indicates the state of the checking result. If the colour is green it means there is no warning or error. If you get warning but no error, the colour will be orange, which means the system can still proceed and the eBook can be read; but with content or definition that is identified as suspicious, the eBook may not read in the way that the author expects it to. A red background message indicates that there is error in the eBook file and it is not recommended to proceed as the eBook will not be opened or read correctly.

At this stage, the author will be asked whether to continue to import the eBook. The options are provided by two buttons as shown in Figure 4-7. If you press "Import", the system will proceed. If you press "Return", the system will abandon the import process and go back to the main page.

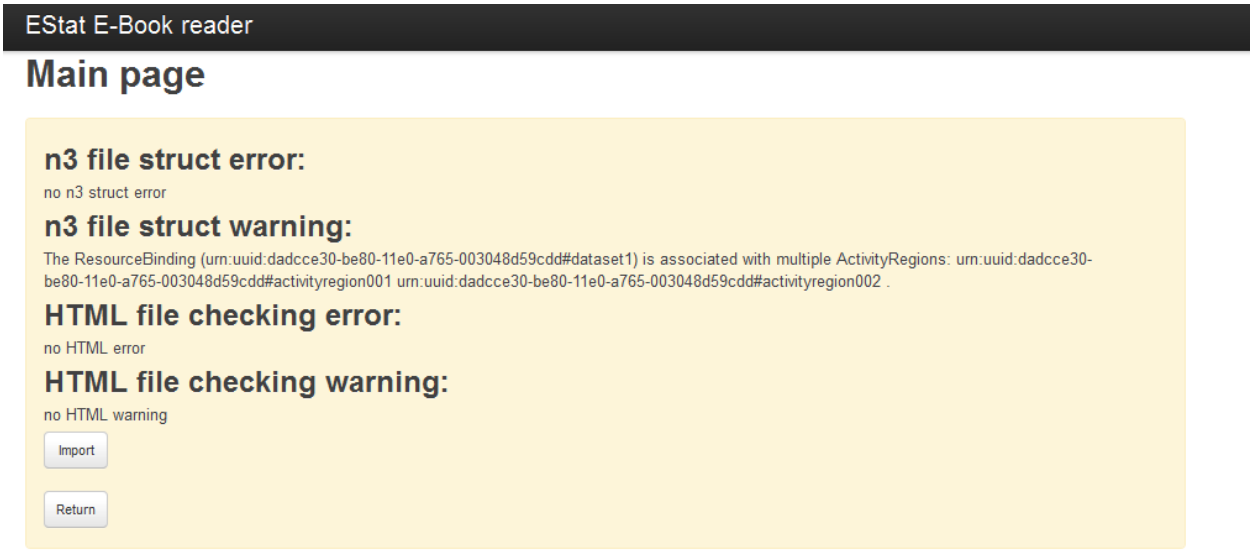


Figure 4-7. File checking result displayed during eBook import

Checking during eBook reloading:

When the user reloads an eBook, the system loads the filechecker to check the consistency and displays the result; the system will then ask the author to decide whether to continue reloading. The options provided by the buttons are shown in Figure 4-8. If "Continue Reloading" is selected, the system will reload the eBook file. If "Cancel Reloading" is selected, the system will abandon the reloading process and go back to the previous reading interface.

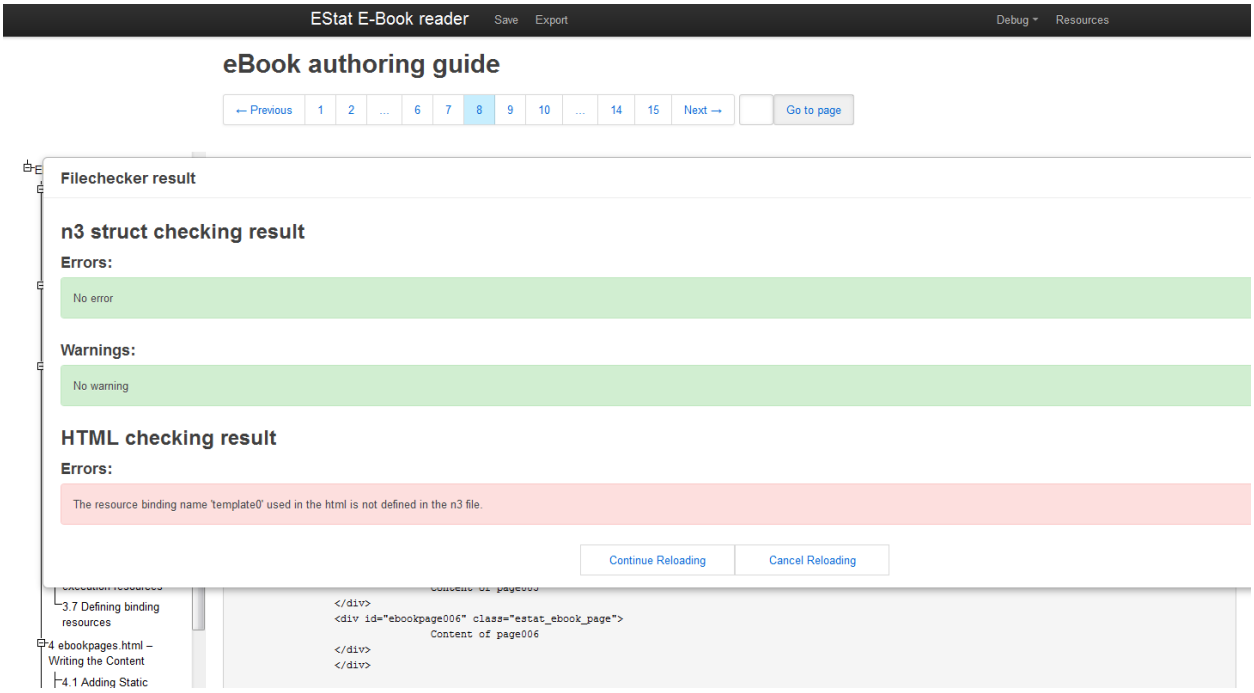


Figure 4-8. File checking result displayed during eBook reloading

Calling checking from the command line:

The user can also call the filechecker from the command line for debugging purposes. The location of the eBook draft's folder will be given as the parameter. To execute the filechecker, in the command line from within the ebooktest directory, type:

python filechecker.py -d draft_ebook_dir_path

The draft eBook directory path starts with '.' if the path is inside and is relative to the ebooktest directory. For example, for an eBook draft folder with name '4' located in the tests directory of the ebooktest system, the command to check it is:

```
python filechecker.py -d ./tests/4/
```

The checking result will be displayed as shown in Figure 4-9. In this example there are several warnings given regarding the ebookdef.ttl file.

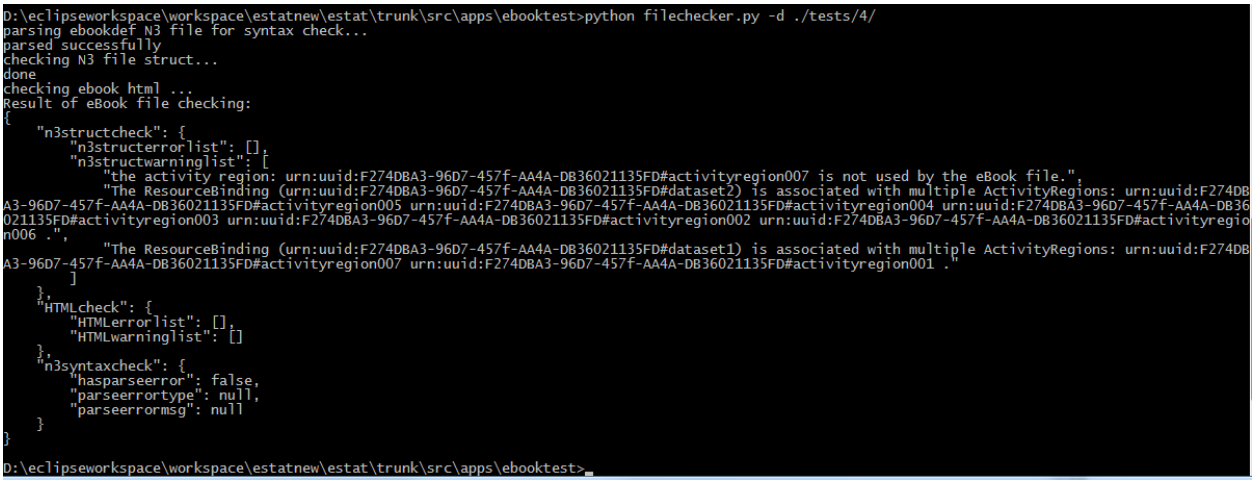


Figure 4-9. File checking result in command line

4.5.3. The Authoring Tab in Resource View

The eBook system provides tools in the authoring tab, which locates in the resource view, to preview the result of your row/column selection or a XPath expression. To use the tools, open the resource and go the authoring panel. Then select the tabular output you want to process in the resource tree on your left hand side. (If you have opened the resource view by clicking on the 'about' button to the right of the tabular output, then it will be selected by default.)

To preview the tabular output result, enter the row/column selections in the 'Only' and 'Except' input boxes and click the Update button. The preview will be displayed underneath the input area, as shown in Figure 4-10.

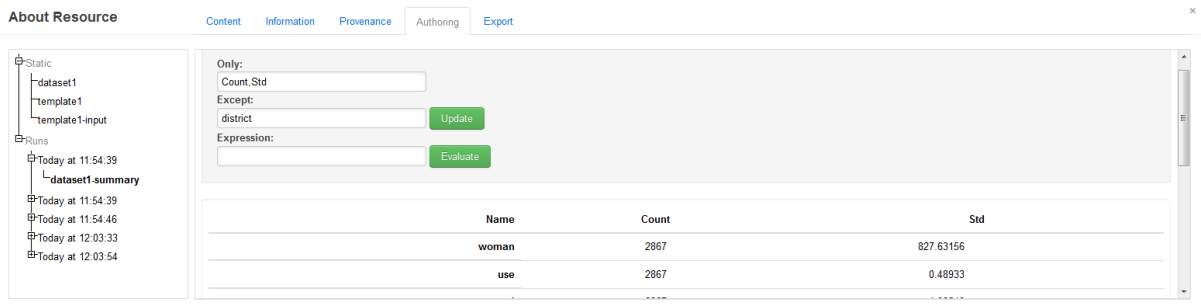


Figure 4-10. Preview result of row/column selection

To evaluate a XPath expression, simply type or paste your it into the 'Expression' input box and click on Evaluate button, the result will be displayed underneath the input area. For example, in the example the data in the row 'woman' and the column 'Count' is 2867, the result of expression `//row[@row='woman']/element[@col='Count' and .>1000]` will be shown as '2867', while the expression `//row[@row='woman']/element[@col='Count' and .<1000]` produces an empty result.

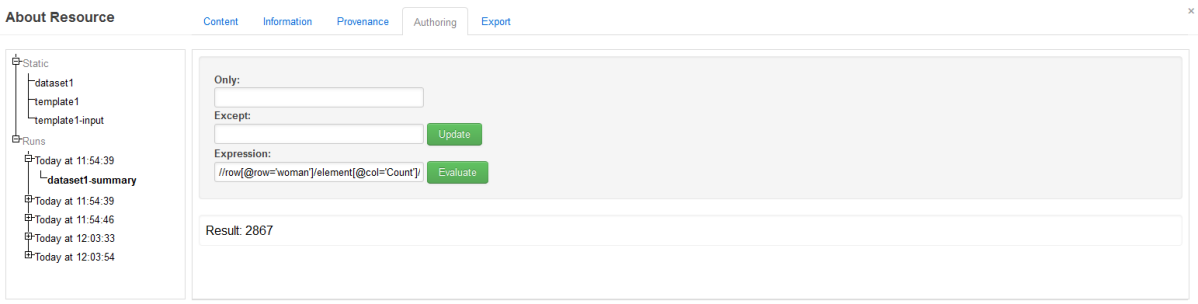


Figure 4-11. Preview result of row/column selection

5 APPENDIX

Appendix I. 'ebookdef.ttl' file for the example eBook described in Section 4.1

```
@base <urn:uuid:dadcce30-be80-11e0-a765-003048d59cdd> .

@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix estat: <http://purl.org/net/statjr/ns#> .
@prefix estatinput: <http://purl.org/net/statjr/input#/> .
@prefix ebook: <http://purl.org/net/deep/ns#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<> a ebook:EbookFile;
    dcterms:created "Tue Jul 02 16:56:47 2011";
    dcterms:creator "Yang, Huanjia";
    dcterms:description "An example of imported eBook.";
    dcterms:language "en";
    dcterms:title "Modelling binary responses";
    ebook:hasActivityRegion <#activityregion001>,
                            <#activityregion002> .

<#activityregion001> a ebook:ActivityRegion;
                     ebook:hasBinding <#template1>,
                                      <#dataset1>,
                                      <#input1> .

<#activityregion002> a ebook:ActivityRegion;
                     ebook:hasBinding <#template2>,
                                      <#dataset1>,
                                      <#input2> .

<#template1> a ebook:ResourceBinding;
             ebook:identifier "template1";
             ebook:resource <#1levelMod> .

<#template2> a ebook:ResourceBinding;
             ebook:identifier "template2";
             ebook:resource <#2levelCat> .

<#dataset1> a ebook:ResourceBinding;
            ebook:identifier "dataset1";
            ebook:resource <#bang> .

<#input1> a ebook:ResourceBinding;
          ebook:identifier "template1-input";
          ebook:resource <#input001> .

<#input2> a ebook:ResourceBinding;
          ebook:identifier "template2-input";
          ebook:resource <#input002> .

<#1levelMod> a estat:Template;
```

```
            ebook:hasName "1LevelMod";
            ebook:isBuiltIn "1LevelMod" .

<#2levelCat> a estat:Template;
            ebook:hasName "2LevelCat";
            ebook:isBuiltIn "2LevelCat" .

<#bang> a estat:Dataset;
        ebook:hasName "Bang";
        ebook:isBuiltIn "bang" .

<#input001> a estat:TemplateInput;
            estatinput:D "Binomial";
            estatinput:y "use";
            estatinput:Engine "eSTAT";
            estatinput:EstM "yes";
            estatinput:outdata "out";
            estatinput:seed "1";
            estatinput:burnin "1000";
            estatinput:iterations "5000";
            estatinput:thinning "1";
            estatinput:link "logit";
            estatinput:n "cons" .

<#input002> a estat:TemplateInput;
            estatinput:L2ID "district";
            estatinput:D "Binomial";
            estatinput:link "logit";
            estatinput:n "cons" ;
            estatinput:y "use";
            estatinput:cons_cat "No";
            estatinput:district_cat "Yes";
            estatinput:age_cat "No";
            estatinput:use_cat "Yes";
            estatinput:use4_cat "Yes";
            estatinput:lc_cat "Yes";
            estatinput:urban_cat "Yes";
            estatinput:educ_cat "Yes";
            estatinput:hindu_cat "Yes";
            estatinput:d_lit_cat "No";
            estatinput:d_pray_cat "No";
            estatinput:outdata "out";
            estatinput:Engine "eSTAT";
            estatinput:EstM "yes";
            estatinput:seed "1";
            estatinput:burnin "1000";
            estatinput:iterations "5000";
            estatinput:thinning "1".
```

Appendix II. An example ‘ebookpages.html’ file

This is an example 'ebookpages.html' file that involves all the functionalities introduced in Chapter 4. The name of dynamic outputs and activity regions are aligned with the 'ebookdef.ttl' file in Appendix I.

```
<div id ="activityregion001" class="deep_activityregion">
<h1>Chapter 1: title of chapter 1</h1><hr>

    <div id ="ebookpage001" class="deep_page">
        <h2>Introduction</h2>
        <p>
            sample text content, paragraphs.... <strong>this is in bold font</strong>.
        </p>
        <div class="deep_static_img">
            <img src="example_static_image.gif" alt="" /><br>
            Figure 1. example static image
        </div>
    </div> <!-- end of ebookpage001-->

    <div data-deep-id="ebookpage002" class="deep_page">
        <h2>Input, Model and Equations</h2>
        <p>
            sample text content, paragraphs....
        </p>
        <p>
        There are inputs to be specified for Stat-JR before model fitting.
        please select (<strong>cons</strong>)and (<strong>age</strong>) in the list below:
        </p>
        <div data-deep-id ="template1-inputq" class="deep_dynamic_output deep_dynamic_hidden">
        </div>
        <p>
        After you submit the input of explanatory variables, you should find that Stat-JR has produced
        below a nicely-formatted mathematical description of the model (in LaTeX code),
        and a variant of the model specification language associated with the WinBUGS package.<br>
        </p>
        <div class="deep_textarea deep_dynamic_hidden">
            <strong>Equation rendering:</strong> (This is the example of static text that shows with
            together with the dynamic content when it becomes available)
            <div data-deep-id="template1-equations" class="deep_dynamic_output
deep_dynamic_hidden">
            </div>
        </div>
        <div id ="textarea_model" class="deep_textarea deep_dynamic_hidden">
            <strong>Model:</strong>
            <div data-deep-id="template1-model" class="deep_dynamic_output deep_dynamic_hidden">
            </div>
        </div>
    </div> <!-- end of ebookpage002-->

    <div id ="ebookpage003" class="deep_page">
        <h2>Results: Summary and Graphs</h2>
        <p>
            Sample content, paragraphs..... The outputs of a model run can be distributed on multiple
            pages, they will be shown as long as they are all in this same activity region.
        </p>
        <div id ="dynamicarea_templsummary" class="deep_textarea deep_dynamic_hidden">
            Here are the summary statistics from running the model:
            <div data-deep-id="template1-summary" class="deep_dynamic_output deep_dynamic_hidden">
            </div>
        </div>

        <div class="deep_dynamic_hidden">
            Here is a graph for deviance.
            <div data-deep-id="template1-graph_deviance" class="deep_dynamic_output
deep_dynamic_hidden">
            </div>
            <p>
```

```
            You can put large portion of text with the dynamic content, as large as you like.
            and you can also nest multiple outputs under one hided div.
            </p>
            <p>
            The deviance formula for a Binomial model is:
            </p>
            You can use static image in the dynamic area as well: <br>
            <div data-deep-id="another_static_image" class="deep_static_img">
                <img src="example_static_image2.gif" alt="" />
            </div>
            <p>
                you can use standard HTML table like this:
            </p>
            <p>
            <table class="deep_static_table" border="1" cellspacing="0" cellpadding="0">
                <tbody>
                <tr>
                <td style="text-align: center;" valign="top" width="136"> </td>
                <td style="text-align: center;" valign="top" width="114">
                <span style="text-decoration: overline;">D</span></td>
                <td style="text-align: center;" valign="top" width="114"> D(
                <span style="text-decoration: overline;">&theta;</span>)</td>
                <td valign="top" width="114"><p align="center"><em>p<sub>D</sub></em></p></td>
                <td valign="top" width="137"><p align="center">DIC</p></td>
                </tr>
                <tr>
                <td valign="top" width="136">Model with age</td>
                <td valign="top" width="114"><p align="right">2591.37</p></td>
                <td valign="top" width="114"><p align="right">2589.29</p></td>
                <td valign="top" width="114"><p align="right">2.07</p></td>
                <td valign="top" width="137"><p align="right">2593.44</p></td>
                </tr>
                <tr>
                <td valign="top" width="136">Model without age</td>
                <td valign="top" width="114"><p align="right">2591.94</p></td>
                <td valign="top" width="114"><p align="right">2590.91</p></td>
                <td valign="top" width="114"><p align="right">1.03</p></td>
                <td valign="top" width="137"><p align="right">2592.96</p></td>
                </tr>
                </tbody>
            </table>
            </p>
            <p>
                And more, you can have headers in the content that goes with the dynamic content,
                They will be added into the table of contents as well when the dynamic content becomes
                available and the whole div becomes visible.
            </p>
            <h3>Graph beta 1</h3>
            <div data-deep-id="template1-graph_beta1"
                 class="deep_dynamic_output deep_dynamic_hidden">
            </div>
            <h3>Graph beta 0</h3>
            <p>
            And here is the <strong> beta 0 </strong> graph result:
            </p>
            <div data-deep-id="template1-graph_beta0"
                 class="deep_dynamic_output deep_dynamic_hidden">
            </div>
        </div>
    </div> <!-- end of ebookpage003-->

</div> <!-- end of activityregion001-->


<!-- Just duplicate the activity region div and change the content  when you want to have more of them, you can have as many activity regions as you want. don't forget the declare them in the n3 file.-->
<div id ="activityregion002" class="deep_activityregion">
<h1>Chapter 2: header of chapter 2</h1>
    <div id ="ebookpage_005" class="deep_page">
    <h2>Introduction, input and model</h2>
        <p>
```

```
        currently, you do not need to declare the pages in the n3.
        </p>
        <div class="deep_dynamic_hidden">
            <p>
            You are not required to give id to the dynamic divs that nest dynamic outputs with their
            accompanying static content. but you can do so for your own convenience.<br>
            Please select <strong>cons</strong>,  <strong>age</strong> and <strong>lc</strong>
            as explanatory variables below.
            </p>
            <div data-deep-id ="template2-inputq" class="deep_dynamic_output deep_dynamic_hidden">
            </div>
        </div>
        <div class="deep_dynamic_hidden">
            <p>
            <strong>Equation rendering:</strong>
            </p>
            <div data-deep-id="template2-equations"
                 class="deep_dynamic_output deep_dynamic_hidden">
            </div>
        </div>
        <p>
    </div> <!-- end of ebookpage_005-->

    <div data-deep-id="I_am_an_ebook_page_name" class="deep_page">
    <h2>Model run summary</h2>
        The system does not care what you give as the id of the page divs', it
        generates page number by counting the pages you defined. for example, you can set the id of
a page as
        'template2-resultpage' if that makes more sense to you.
            <p>
            Once the model has run, as well as noting the change in the coefficient
            of age, we see that the DIC is considerably smaller.
            </p>
            <div data-deep-id="template2-summary" class="deep_dynamic_output deep_dynamic_hidden">
            </div>
            <p>
    </div> <!-- end of I_am_an_ebook_page_name -->

    <div id="ebookpage_006" class="deep_page">
    <h2>Results</h2>
         You can use Latex in your content and they will be rendered:
            <p>
            \[ normexam_i \sim N(X\beta, \Omega) \]
            \[ normexam_i = \beta_{0i} cons + \beta_{1i} standlrt_{i} \]
            This is the model written in bugs model specification language:
            </p>
            <div data-deep-id="template2-model" class="deep_dynamic_output deep_dynamic_hidden">
            </div>
            <p>
            Some more explanation can be provided here for the model generated.
            </p>

        <div class="deep_dynamic_hidden">
            <p>
            Here is a graph for beta0.
            </p>
            <div data-deep-id="template2-graph_beta0"
                 class="deep_dynamic_output deep_dynamic_hidden">
            </div>
            <p>
            Some more explanation can be provided here for the graph generated.
            </p>
        </div>
    </div> <!-- end of ebookpage_006 -->
</div> <!-- end of activityregion002 -->
```

Appendix III. List of HTML Markers Introduced in eBook System

Class type introduced	**(to be used as value of a HTML element's "class" attribute)**
deep_activityregion	Mark a div as an eBook activity region
deep_page	Mark a div as an eBook page
deep_dynamic_output	Mark an element as a dynamic output
deep_dynamic_hidden	Mark an element as invisible
deep_static_img	Mark an image element to be displayed with format pre-defined by eBook system
Attribute introduced	**(to be used as an attribute of a HTML element)**
data-deep-showon= *<dynamic_output>*	Mark an element to be shown when the dynamic output specified in the value of the attribute becomes available.
data-deep-hideon= *<dynamic_output>*	Mark an element to be hidden when the dynamic output specified in the value of the attribute becomes available.
data-deep-only= *<row/column names>*	Specify the rows/columns to be displayed for a dynamic output
data-deep-except= *<row/column names>*	Specify the rows/columns to be excluded when displaying a dynamic output
data-deep-expression= *<XPath_expr>*	Specify a XPath expression for accessing data value in dynamic output